NEUROMORPHIC REINFORCEMENT LEARNING FOR QUADRUPED LOCOMOTION CONTROL ON UNEVEN TERRAIN

Anonymous authors
Paper under double-blind review

ABSTRACT

Most learned quadruped controllers are trained offline with backpropagation and deployed with fixed policy parameters. This paradigm has produced robust locomotion, but future long-duration autonomy will require robots to learn from real-world experience under onboard power and memory constraints. Local learning rules offer an approach to such on-robot adaptation because they replace global backpropagation graphs with synaptic updates driven by local neural states, which is compatible with low-power neuromorphic devices. A prerequisite for this objective is to demonstrate that local learning is capable of training a controller for robotic locomotion tasks. We investigate this feasibility question using equilibrium propagation (EP) to train both policy and value networks in a simulated uneven-terrain environment. We develop an EP-based proximal policy optimization (PPO) framework within an established central pattern generator and residual postural control architecture, and we construct a PPO-derived nudging rule with a per-relaxation-step ratio gating, two-sided clipping, and modified gradient scaling. Experiments on a simulated 12-DoF Unitree A1 quadruped demonstrate successful two-stage locomotion training. The resulting controller achieves comparable walking performance to a backpropagation-trained PPO baseline on the reported metrics. The EP-based policy update rule reports approximately $4.3\times$ lower GPU memory consumption for EP updates than for backpropagation through time (BPTT). These results establish the algorithmic feasibility of EP-based local learning for quadruped locomotion training and provide a foundation for future low-power on-robot adaptation.

1 INTRODUCTION

Quadruped locomotion has progressed from laboratory demonstrations on flat ground to robust traversal of irregular, deformable, and cluttered terrain. This progress has been driven by controllers that coordinate cyclic stance–swing leg motions to maintain stability, generate propulsion, and recover from disturbances. Model predictive control (MPC) and RL have enabled dynamic locomotion, recovery, and agile navigation across multiple quadruped platforms Kim et al. (2019); Hwangbo et al. (2019); Hoeller et al. (2024); Kumar et al. (2021); Smith et al. (2022). These advances support applications in industrial inspection, logistics, disaster response, and remote operation. However, high locomotion performance alone is insufficient for long-duration autonomy: the computational energy required by the controller becomes an increasingly important constraint.

Most high-performing RL locomotion controllers are trained off-robot in contact-rich simulation and transferred to hardware through domain randomization, system identification, actuator modeling, or related sim-to-real techniques Hwangbo et al. (2019); Margolis et al. (2024); Tan et al. (2018). Their policy parameters are then held fixed during deployment. Simulation-trained adaptation modules can adjust behavior online by inferring latent environment or robot properties Kumar et al. (2021), but they do not update the locomotion policy from new deployment experience. When a robot encounters conditions that are not adequately represented during training, including new terrain statistics, payload changes, actuator degradation, or accumulated mechanical wear, a fixed-parameter controller may require data collection and retraining on external computing infrastructure.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Learning directly from deployment experience could reduce dependence on this simulation–deployment–retraining cycle. Prior work has demonstrated real-world robot learning, but such systems still face severe constraints in data rate, safety, supervision, and available computation Haarnoja et al. (2018); Ha et al. (2020); Smith et al. (2022); Bloesch et al. (2022). These constraints are especially important for long-duration or remote operation, where a robot cannot rely on continuous access to external machine learning accelerators or power supplies. This motivates algorithm–hardware co-design for low-power locomotion learning, where the learning rule itself should be compatible with efficient physical implementation.

Biological locomotor systems offer useful inspiration for this goal. Animals do not command each muscle independently; instead, high-level motor centers modulate spinal circuits that generate rhythmic locomotor patterns, while sensory feedback and reflex pathways adapt these patterns during interaction with the environment Ijspeert (2008); Righetti & Ijspeert (2008); Kimura et al. (2007); Manoonpong et al. (2013); Kiehn (2006; 2016); Grillner & El Manira (2020); Grillner & Kozlov (2021). CPG-based robotic controllers exploit this organization by converting actuator-level control into structured rhythmic parameters such as phase, amplitude, frequency, and orientation. In RL-based locomotion, this structure can reduce the burden on the learned policy by constraining the action space toward meaningful gait patterns Ijspeert (2008); Bellegarda & Ijspeert (2022); Lee et al. (2020); Suzuki et al. (2025). However, CPGs organize limb-level motion but do not by themselves solve the computational and learning-efficiency challenges of trainable neural controllers. This motivates a dynamical-systems perspective in which oscillator dynamics structure locomotion control, while neural relaxation dynamics provide a mechanism for local learning and adaptation.

This remaining challenge connects naturally to Neuro-AI, which seeks to use principles from biological intelligence to develop artificial agents with stronger robustness, adaptation, and energy efficiency. For embodied robots, this perspective shifts the focus from only accelerating inference to designing learning rules that can adapt under local information and limited onboard resources. Backpropagation remains the dominant training method for deep RL, but it relies on global credit assignment, activation storage, and backward gradient transport, which are difficult to reconcile with biological or physical learning substrates. Recent high-impact alternatives, including feedback alignment, target propagation, predictive-coding learning, and equilibrium propagation Lillicrap et al. (2016); Lee et al. (2015); Whittington & Bogacz (2017); Scellier & Bengio (2017), show that multilayer network learning can be approximated or implemented through local activity, target states, or dynamical perturbations rather than exact backpropagation.

Among these approaches, EP is especially relevant because both inference and learning are expressed through network relaxation dynamics. EP trains energy-based networks by comparing nearby equilibrium states induced by small output nudges, avoiding explicit global backpropagation graphs and aligning naturally with in-memory and neuromorphic hardware dynamics Scellier & Bengio (2017); Laborieux et al. (2021). Recent EP variants further suggest its potential for efficient local learning Laborieux et al. (2021); Laborieux & Zenke (2022; 2023); Scellier et al. (2023); Martin et al. (2021), but EP has mainly been studied on supervised benchmarks. Its use in reinforcement learning remains comparatively limited. To the best of our knowledge, prior work has not demonstrated EP-based training of both continuous-action policy networks and value networks for quadruped locomotion over randomized uneven terrain. Establishing this capability is the central objective of this paper.

Motivated by the eventual goal of low-power on-robot adaptation, the central contribution of this work is to demonstrate that EP-based local learning can train a quadruped locomotion controller. This feasibility result is a prerequisite for future systems that update locomotion policies from deployment experience using local neuromorphic learning mechanisms. We investigate this question within the established CPG–RES architecture of Seto et al. (2025), which builds on the CPG-RL controller of Bellegarda & Ijspeert (2022). This architecture provides a structured locomotion task for evaluating the learning algorithm. Our contributions are:

1. Demonstrating the feasibility of EP-based quadruped locomotion training. We train both policy and value networks using EP-based updates in a two-stage locomotion task for a simulated 12-DoF Unitree A1. Training and walking evaluations across five independent seeds show that EP-based local learning can produce a functional uneven-terrain locomotion controller with performance broadly comparable to the BP-PPO baseline on the reported metrics.

2. Ratio-gated EP nudging for continuous-action policies. We recompute the PPO likelihood ratio from the evolving EP output state at each nudged relaxation step and use a two-sided clipping mask to gate the output-layer force. This within-relaxation feedback adapts PPO clipping to EP dynamics; modified gradient scaling further stabilizes training. The ablations characterize the resulting training configuration.

2 Related Works

Biological locomotion relies on hierarchical motor organization: high-level commands modulate rhythmic spinal circuits, while sensory feedback and reflex pathways adapt movement during environment interaction Kiehn (2006; 2016); Grillner & El Manira (2020); Grillner & Kozlov (2021). This principle has inspired CPG-based robotic controllers, where oscillatory dynamics generate coordinated periodic limb trajectories and reduce the dimensionality of locomotion control. Classic CPG studies demonstrate stability, adaptability, and sensory-entrainment benefits for legged robots Ijspeert (2008); Righetti & Ijspeert (2008); Kimura et al. (2007); Manoonpong et al. (2013). In RL-based locomotion, CPGs constrain policy search toward structured rhythmic movements instead of unconstrained joint torques or joint targets. Prior work has combined CPGs with RL for hexapod and quadruped locomotion, including PPO-based CPG-RL, visually guided CPG locomotion, residual postural policies for uneven terrain, and foot-trajectory studies Ouyang et al. (2021); Bellegarda & Ijspeert (2022); Bellegarda et al. (2024); Seto et al. (2025); Suzuki et al. (2025). These studies show that CPG architectures provide useful locomotion priors, but their trainable components are still generally optimized by backpropagation, limiting direct deployment on local-learning neuromorphic substrates.

Local learning algorithms provide a more biologically plausible approach for resource-constrained learning. Feedback alignment relaxes the weight-transport requirement of backpropagation, target propagation replaces layerwise gradients with local target states, predictive-coding networks approximate backpropagation through iterative inference and local Hebbian updates, and EP formulates learning as the contrast between free and weakly nudged equilibrium states Lillicrap et al. (2016); Lee et al. (2015); Whittington & Bogacz (2017); Scellier & Bengio (2017). These methods support the Neuro-AI goal of adaptive, biologically inspired learning systems, but most have been evaluated primarily on supervised benchmarks rather than contact-rich robot learning.

Recent EP developments have improved scalability and hardware relevance through second nudging phases, holomorphic formulations, Jacobian homeostasis, asynchronous relaxation, analog-computing analyses, and spike-driven implementations Laborieux et al. (2021); Laborieux & Zenke (2022; 2023); Scellier et al. (2023); Martin et al. (2021). Nevertheless, EP remains underexplored in reinforcement learning. Prior actor–critic work trained an actor with EP while retaining a backpropagation-trained critic Kubo et al. (2022). Here, both policy and value networks use EP; for the continuous-action policy, the PPO likelihood ratio is recomputed as the output state relaxes and gates the nudging term at each step. We evaluate this training rule within the established CPG–RES controller on uneven-terrain locomotion.

Together, these lines of work motivate a controller that combines the structured locomotion priors of CPG-based control with local neuromorphic learning in the trainable policy and value networks. The following section describes the proposed EP-based CPG–RES quadruped controller and its PPO training procedure.

3 Methods

We use a two-branch CPG–RES controller to combine structured rhythmic gait generation with learned postural correction. The controller is evaluated on a 12-DoF Unitree A1 quadruped simulated in MuJoCo. Each limb contains hip, thigh, and calf joints; we use $i \in \{1, 2, 3, 4\}$ to index limbs and $j \in \{\mathrm{hip}, \mathrm{thigh}, \mathrm{calf}\}$ to index joints within a limb.

The reinforcement learning task of this paper is to drive a quadruped robot to walk robustly on uneven terrain. Several RL-based training paradigms have achieved outstanding locomotion stability and agility on various types of quadruped platforms Kumar et al. (2021); Hwangbo et al. (2019); Margolis et al. (2024). The control architecture in this paper follows the CPG-RES two-stage learn-

ing paradigm in Seto et al. (2025). Fig. 1 shows a schematic of the control architecture. This architecture contains two RL policy networks: the CPG policy network and the RES policy network. At each RL control timestep, these two policy networks map the observation vector to two action vectors: the CPG oscillator parameter vector and the residual angle derivative vector, respectively. For the CPG branch, the CPG oscillator parameter vector is fed into the CPG oscillator. At each low-level time step, the CPG oscillator provides three CPG states (r_i, θ_i, ϕ_i). The (r_i, θ_i, ϕ_i) controls a foot trajectory generator, which outputs CPG foot Cartesian coordinates. The CPG foot Cartesian coordinates are converted into CPG joint angles through inverse kinematics. For the residual angle branch, integrating the residual angle derivatives gives the residual angles. The residual angles are added to the CPG joint angles, and the resulting target joint angles are converted into the joint torques through a PD controller. Joint torque is the lowest-level command for controlling a robot in both simulation and the real world.

3.1 EP-BASED CPG–RES LOCOMOTION CONTROLLER

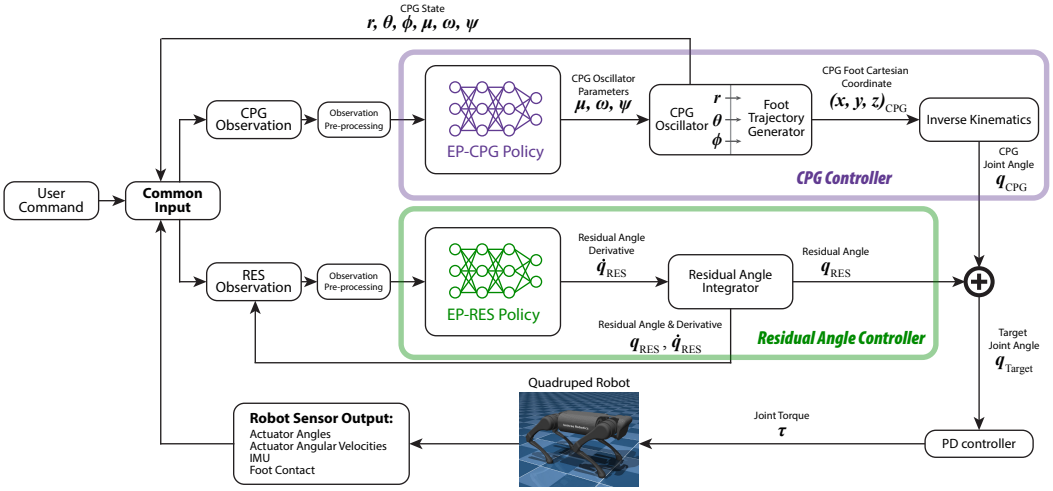

Figure 1: The control architecture of the EP-based quadruped locomotion control. This schematic shows the stage-2 uneven terrain control iteration. The training process is not shown here. The overall control architecture follows Seto et al. (2025) with minor modifications.

The details of the CPG oscillator, foot trajectory generator, and the residual angle integration are reported in Appendix A. In brief, after the CPG controller and the residual angle controller output the CPG joint angle and the residual angle, respectively, the overall target joint angle is the sum of their outputs:

$$q_{j,i,\text{target}} = q_{j,i,\text{CPG}} + q_{j,i,\text{RES}} \quad \forall j, i \tag{1}$$

Here $q_{j,i,\text{CPG}}$ is the CPG joint angle and $q_{j,i,\text{RES}}(t)$ is the residual angle. The PD controller has the following form:

$$\tau_{j,i} = \mathbf{Clamp}(-K_p(q_{j,i} - q_{j,i,\text{target}}) - K_d\,\dot{q}_{j,i}, -\mathbf{TorqueLimit}, \mathbf{TorqueLimit}) \quad \forall j, i \tag{2}$$

Here, $\tau_{j,i}$, $q_{j,i}$, $q_{j,i,\text{target}}$, and $\dot{q}_{j,i}$ are joint torque to be applied, current joint angle, target joint angle, current angular velocity of joint j at limb i, respectively. Two PD hyper-parameters are $K_p = 100\,\text{N m}$ and $K_d = 2\,\text{N m s}$. Also, all the calculated joint torques are clamped to the actuator torque range of $[-\mathbf{TorqueLimit}, \mathbf{TorqueLimit}]$ before being applied to the MuJoCo environment.

The learned CPG and RES policies run at 100 Hz, while the CPG oscillator, residual-angle integrator, PD controller, and MuJoCo simulation run at 1000 Hz. Thus, each high-level policy action is held for 10 low-level control steps. The CPG and RES policies receive 63-D and 87-D observations, respectively, and each outputs a 12-D action vector. Details about the robot parameters, oscillator dynamics, foot trajectory generator, residual integrator dynamics, observation/action definitions, reward function, and hyper-parameters are given in Appendix A and G. Readers are also directed to Seto et al. (2025) for the original proposal of the CPG–RES architecture.

3.2 EP-PPO FOR CONTINUOUS-CONTROL LEARNING

We use PPO Schulman et al. (2017) as the RL framework and train the policy and value networks through EP-based updates. The CPG policy is trained in stage 1. In stage 2, the CPG policy is fixed while the RES policy is trained. In each stage, the trainable policy and value networks use local neuronal-state-based updates without backpropagation through those networks.

To specify the PPO data used for the EP updates, consider one environment first. At each control step t, its rollout buffer stores the tuple

$$(s_t, a_t, r_t, \log(\pi_{\text{rollout}}(a_t|s_t)), d_t, \text{fall}_t, \mathbf{Value}(s_t), \mathbf{Value}(s_{t+1}))$$

Here, s_t, a_t, and r_t denote the observation vector, action vector, and scalar reward. $\pi_{\text{rollout}}(a_t|s_t)$ is the action probability calculated in the rollout before any policy update, and "rollout" is used to distinguish the before-policy-update probability and after-policy-update probability. d_t is the episode ending signal (see Appendix A for episode ending conditions), and fall_t is the robot falling signal. $\mathbf{Value}(s_t), \mathbf{Value}(s_{t+1})$ are the value function of s_t and s_{t+1}. In parallel simulation framework considered herein, there are N parallel environments, and, in a single rollout, each environment forwards T control steps. A total of $N \cdot T$ data tuples will be collected in the rollout phase. In the neural network update phase, the data tuples stored in the buffer are processed. $\forall t$, first, $\mathbf{Return}(s_t)$ and $\mathbf{Advantage}(s_t, a_t)$ are calculated through generalized advantage estimator (GAE) Schulman et al. (2015), which requires r_t, d_t, fall_t and $\mathbf{Value}(s_t), \mathbf{Value}(s_{t+1})$. Subsequently, the policy network and the value network are updated for K_{epoch} epochs and $\frac{|\mathcal{D}|}{|\mathcal{B}|}$ mini-batches in each epoch, where $\mathcal{D}$ is a single rollout dataset with $N \cdot T$ data tuples, and $\mathcal{B}$ is a data mini-batch. Training the value network is formulated as a regression task. See details about the value loss function in Appendix C.

The policy network is updated by maximizing the PPO action surrogate objective function:

$$L^{\text{CLIP}} = \hat{\mathbb{E}}_t[\min(r_t(\mu_t)\hat{A}_t, \mathbf{Clamp}(r_t(\mu_t), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3}$$

Here, $\hat{A}_t$ is $\mathbf{Advantage}(s_t, a_t)$, and ϵ is the PPO ratio clip coefficient. $r_t(\mu_t) = \frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)}$ is the probability ratio. $\pi(a_t|s_t)$ is the current-policy probability of sampling the rollout action a_t given observation s_t. If $\mu_{t,i}$ is the i-th dimension of the policy output action mean in sample t, given the PPO assumption $r_t(\mu_t) \approx 1$ in the update process, we have the approximate PPO objective gradient w.r.t. the policy network output $\mu_{t,i}$:

$$\begin{aligned}\frac{\partial L^{\text{CLIP}}}{\partial \mu_{t,i}} &= \frac{1}{|\mathcal{B}|}\begin{cases}\mathbf{1}_{[0,1+\epsilon)}(r_t(\mu_t)) \cdot r_t(\mu_t)\frac{(a_{t,i}-\mu_{t,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \mathbf{1}_{(1-\epsilon,\infty)}(r_t(\mu_t)) \cdot r_t(\mu_t)\frac{(a_{t,i}-\mu_{t,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t < 0\end{cases} \\ &\approx \frac{1}{|\mathcal{B}|}\begin{cases}\mathbf{1}_{[0,1+\epsilon)}(r_t(\mu_t)) \cdot \frac{(a_{t,i}-\mu_{t,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \mathbf{1}_{(1-\epsilon,\infty)}(r_t(\mu_t)) \cdot \frac{(a_{t,i}-\mu_{t,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t < 0\end{cases}\end{aligned} \tag{4}$$

Here, the terms $\mathbf{1}_{[0,1+\epsilon)}(r_t(\mu_t))$ and $\mathbf{1}_{(1-\epsilon,\infty)}(r_t(\mu_t))$ can be regarded as gradient masks. It sets the continuous part of the gradient $\frac{(a_{t,i}-\mu_{t,i})}{\sigma_i^2}\hat{A}_t$ to zero when the probability ratio is beyond the gradient enabling range (positive advantage: $[0, 1+\epsilon)$, negative advantage: $(1-\epsilon, \infty)$). For the full derivation of the PPO objective gradient, see Appendix C.

Beyond the PPO objective gradient, it is important to understand how the EP network optimizes a given objective function. The EP network is a dynamic system, where all the neuron states evolve according to the negative energy gradient. The energy function of EP has the following form when the task is to minimize a loss function L:

$$E(\xi) = \frac{1}{2}\sum_i \xi_i^2 - \sum_{i,j} w_{ij}\rho(\xi_i)\rho(\xi_j) - \sum_i b_i\rho(\xi_i) - \beta L(\xi_{out}, y) \tag{5}$$

Here, ξ_i is the state of neuron i. $\rho(\cdot)$ is the activation function. w_{ij} and b_i are the weights and biases. $L(\xi_{out}, y)$ is the loss function. ξ_{out} is the output layer neuron's state vector. β is the EP nudge coefficient. β determines whether the loss function influences the neuron states. Note that the β sign convention is different than the original formulation in Scellier & Bengio (2017). The EP network

converges to an equilibrium following the dynamics. For neuron i in the output layer $\xi_{out,i}$, the time derivative is:

$$\frac{d\xi_{out,i}}{d[\text{time}]} = -\frac{\partial E}{\partial \xi_{out,i}} = -\xi_{out,i} + \rho'(\xi_{out,i}) \sum_j w_{out,ij}\rho(\xi_j) + b_{out,i}\rho'(\xi_{out,i}) + \beta \frac{\partial L(\xi_{out})}{\partial \xi_{out,i}} \quad (6)$$

For three-phase EP, the network first reaches a free equilibrium at $\beta = 0$. The free phase equilibrium neuron state is denoted by ξ^*. Then the EP network reaches two nudge equilibrium states at $\beta = \beta_{\text{ep}}$ and $\beta = -\beta_{\text{ep}}$. The initial state value for the nudge phase relaxation is ξ^*. The gradient of the loss function w.r.t. the EP network parameters w_{ij} (a weight matrix element) can be approximated by using the two nudge equilibrium states, $\forall i, j$

$$\frac{\partial L(\xi^*_{out}, y)}{\partial w_{ij}} \approx \frac{1}{2\beta_{\text{ep}}}[\rho(\xi_i^{+\beta_{\text{ep}}})\rho(\xi_j^{+\beta_{\text{ep}}}) - \rho(\xi_i^{-\beta_{\text{ep}}})\rho(\xi_j^{-\beta_{\text{ep}}})] \quad (7)$$

Here, $\xi_i^{+\beta_{\text{ep}}}$ and $\xi_i^{-\beta_{\text{ep}}}$ are the equilibrium states at of the positive and negative nudge phases. For a bias parameter b_i, the form of loss gradient is similar:

$$\frac{\partial L(\xi^*_{out}, y)}{\partial b_i} \approx \frac{1}{2\beta_{\text{ep}}}[\rho(\xi_i^{+\beta_{\text{ep}}}) - \rho(\xi_i^{-\beta_{\text{ep}}})] \quad (8)$$

It can be summarized that, by applying $\beta \frac{\partial L(\xi_{out}, y)}{\partial \xi_{out}}$ at the output layer neurons in the nudge phases, the EP learning mechanism can calculate the loss gradient w.r.t. the network parameters through equilibrium differentiation. Therefore, to optimize the PPO objective, we can apply the PPO objective gradient w.r.t. the network output $\frac{\partial L^{\text{CLIP}}}{\partial \mu_{t,i}}$ to the output layer neurons in the EP network. Note that there is no neuron state dynamics or free-nudge relaxation in the traditional backpropagation-trained feed-forward network – there is only a single output layer activation μ_t without any temporal dynamics. The $\mu_{t,i}$ is the policy network output dimension i, and is equivalent to $\xi^*_{t,out,i}$ in EP, which is the output neuron state i in free phase equilibrium when the policy input observation is s_t.

By default, the EP network wants to minimize a loss function $L(\xi_{out}, y)$. However, here the PPO algorithm requires its objective to be maximized. Therefore, we can apply the negative of the PPO objective gradient w.r.t. the network output, shown in Equation 4, directly to the EP output layer, which yields an output layer dynamics:

$$\begin{aligned}\frac{d\xi_{t,out,i}}{d[\text{time}]} =& -\xi_{t,out,i} + \rho'(\xi_{t,out,i}) \sum_j w_{out,ij}\rho(\xi_{t,j}) + b_{out,i}\rho'(\xi_{t,out,i}) \\ & - \beta \cdot \frac{1}{|\mathcal{B}|} \begin{cases} \mathbf{1}_{[0,1+\epsilon)}(r_{t,\text{nudging}}(\xi_{t,out})) \cdot \frac{(a_{t,i} - \xi_{t,out,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \mathbf{1}_{(1-\epsilon,\infty)}(r_{t,\text{nudging}}(\xi_{t,out})) \cdot \frac{(a_{t,i} - \xi_{t,out,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t < 0 \end{cases}\end{aligned} \quad (9)$$

Here $r_{t,\text{nudging}}(\xi_{t,out})$ is the nudging probability ratio, which is dedicated to the nudge phase per-relaxation-iteration probability ratio calculation:

$$\begin{aligned} r_{t,\text{nudging}}(\xi_{t,out}) &= \frac{\pi_{\text{nudging}}(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)} \\ &= \frac{1}{\pi_{\text{rollout}}(a_t|s_t)} \prod_i^{D_{\text{action}}} \left[\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(a_{t,i} - \xi_{t,out,i})^2}{2\sigma_i^2} \right) \right] \end{aligned} \quad (10)$$

Here $\pi_{\text{nudging}}(a_t|s_t)$ is the action probability in the nudge relaxation as a function of $\xi_{t,out,i}$. D_{action} is the dimensionality of action space. The reason for using the notation $r_{t,\text{nudging}}(\xi_{t,out})$ instead of $r_t(\xi_{t,out})$ is that the probability ratio keeps changing/oscillating in both nudge phase iterations because $\xi_{t,out}$ keeps changing/oscillating, and this nudging probability ratio controls the gradient mask in each relaxation step. However, experiments show that this formulation fails to converge when using Equation 9 as the objective gradient. The per-update KL-divergence graph indicates excessively large update steps. Upon investigation of the cause of the large update steps, we conclude that, for positive-advantage samples, the positive nudge phase will drive the output neuron state toward the **farther-away-from-target** direction without bound. For a detailed discussion about

the cause of the large update step using the original PPO objective gradient, see Appendix C. To constrain these large update steps, we propose the two-sided PPO ratio clip objective gradient:

$$\frac{\partial L^{\text{TwoSidedCLIP}}}{\partial \xi_{t,out,i}} = \frac{1}{|\mathcal{B}|} \begin{cases} \mathbf{1}_{(1-\epsilon_{\text{rev}},1+\epsilon)}(r_{t,\text{nudging}}(\xi_{t,out})) \cdot \frac{(a_{t,i}-\xi_{t,out,i})}{\sigma_i} \hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \mathbf{1}_{(1-\epsilon,1+\epsilon_{\text{rev}})}(r_{t,\text{nudging}}(\xi_{t,out})) \cdot \frac{(a_{t,i}-\xi_{t,out,i})}{\sigma_i} \hat{A}_t & \text{if } \hat{A}_t < 0 \end{cases} \tag{11}$$

Here, ϵ_{rev} is the reverse clipping coefficient, and $1/\sigma_i$ replaces the original $1/\sigma_i^2$ scaling. It is worth mentioning here that $r^{(k)}_{t,\text{nudging}} = r_{t,\text{nudging}}(\xi^{(k)}_{t,out})$ is temporally varying with the evolving output state in the nudge phase (where k indexes relaxation steps within a nudge phase, distinct from rollout index t). The two-sided indicator therefore gates the nudging term at each step, while $(a_{t,i}-\xi^{(k)}_{t,out,i})$ changes its magnitude and direction. BP-PPO has no corresponding within-relaxation feedback because its policy network does not undergo a nudged relaxation phase. The experiments show that this two-sided bounded gradient and reduced gradient scaling factor $\frac{1}{\sigma_i}$ successfully leads to training convergence. The ablation studies regarding different values of ϵ_{rev} and $\frac{1}{\sigma_i}$ are shown in Appendix E. The additional experiments in Appendix D show the advantage-r_{nudging} relationship and explain the policy collapse when $\epsilon_{\text{rev}} \geq 1.0$.

Because EP inference requires iterative relaxation, reducing the number of relaxation steps is important for both training time and control latency. We therefore apply an inverse discrete cosine transform (iDCT)-based observation preprocessing step before the EP networks. Empirically, this preprocessing accelerates convergence to equilibrium and reduces the required inference iterations. The ablation study in Appendix E evaluates its effect.

Full derivations of the Gaussian policy gradient and the log-std update are provided in Appendix C.

3.3 Locomotion training paradigm

Training follows the two-stage paradigm of the CPG–RES controller. In stage 1, only the CPG policy is trained on flat ground to learn stable forward locomotion and velocity tracking. In stage 2, the trained CPG policy is fixed, and the RES policy is trained on randomized uneven terrain to provide joint-level postural corrections. The reward combines forward-velocity tracking, lateral/yaw tracking regularization, body-stability penalties, and actuator-power regularization. Rollouts are collected from parallel MuJoCo environments. EP-PPO algorithm pseudo-code is provided in Appendix F. Detailed hyper-parameters, domain randomization ranges, terrain generation, robot initialization, observation preprocessing, and EP network architecture/setup are provided in Appendices B, C, and G.

4 Experiments

The experiments evaluate whether EP-based local learning can successfully train a quadruped locomotion controller and how the resulting behavior compares with a conventional BP-PPO reference. We examine training convergence, learned walking performance, and the effects of the proposed nudging modifications. The BP-PPO comparison uses the same locomotion task, CPG–RES control architecture, observation and action definitions, reward, and rollout budget. EP and BP differ in their neural architectures, update rules, and preprocessing: EP uses recurrent relaxation with iDCT-expanded observations, whereas the feed-forward BP baseline uses normalized observations without iDCT. The foot-trajectory and environment settings used for evaluation are specified in Appendix B.

The training experiments were conducted using up to two NVIDIA RTX A5000 GPUs. Each GPU has 24,564 MB of memory. The CPU memory consumption by stage-1 and stage-2 is approximately 60 GB and 200 GB, respectively. The massively parallel MuJoCo environment consumes the majority of the CPU memory.

4.1 Training convergence and sample efficiency

Sample efficiency is an important criterion for RL algorithms. BP version and EP version of PPO are used in both stage-1 and stage-2 training. Both stages are trained for 10^8 samples. For both EP and BP, we trained 5 independent run versions through random seeding. Note that before training

an RES policy network in stage-2, a CPG policy must be trained in advance and loaded into the stage-2 training program. Each version contains a CPG policy and an RES policy that are trained sequentially, i.e., the RES policy of version 1 is trained based on the CPG policy of version 1, and the RES policy of version 2 is trained based on the CPG policy of version 2, and so on. Fig. 2 depicts the variation of rollout average per-step reward versus the number of training samples. The center line and the shaded region indicate the mean $\pm$ standard deviation over the 5 versions. Fig. 2 (a),(c) shows that, in both stages, EP and BP reach similar final reward levels, and EP can stably improve the training reward without policy collapse. In stage-1, EP and BP converge with similar sample complexity. Also, through Fig. 2 (b),(d), the value MSE versus training samples indicates that both EP and BP value networks can continuously minimize the value loss. The value MSE is defined as the mean squared error between the value function output and the rollout return.

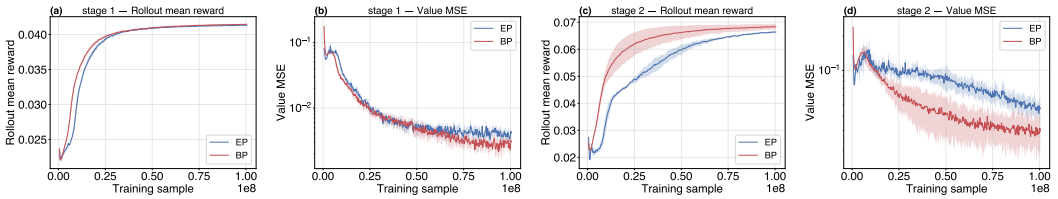

Figure 2: The convergence process of the locomotion training of BP and EP for stage-1 and stage-2. (a) and (b) shows the stage-1 rollout mean reward and MSE error between the value network output and the rollout return. (c) and (d) shows the same metric for the stage-2 training.

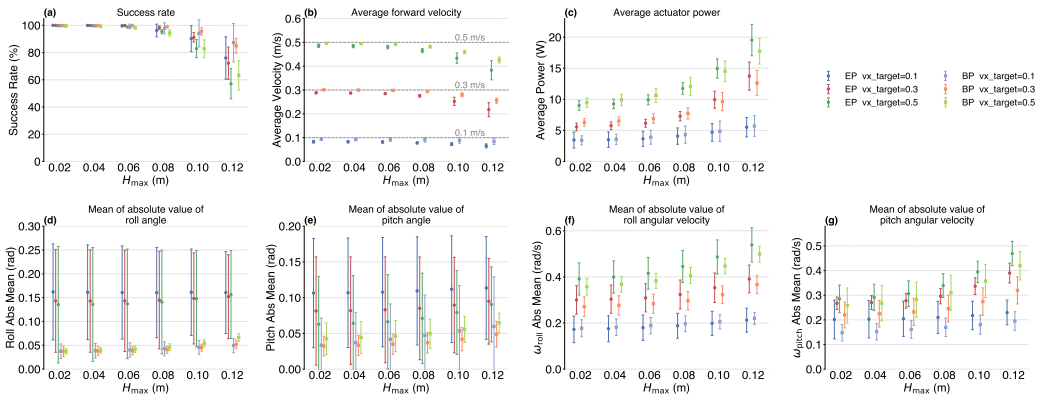

Figure 3: Locomotion performance measured in the walking test. From (a) to (g), the indicators are success rate, average forward velocity, average actuator total power, average absolute roll/pitch angle, and average absolute roll/pitch angular velocity. The center line and the error bar indicate the mean $\pm$ standard deviation over the 5 versions.

4.2 Locomotion performance on uneven terrain

We use a walking test, similar to the 5-meter walking test in Seto et al. (2025), to measure the capability of the control policy to maintain the robot's stability when traveling across obstacles on uneven terrain. On the uneven terrain, both CPG and RES policies are enabled. The following locomotion performance indicators are recorded: success rate, average forward velocity, average actuator total power, average absolute roll/pitch angle, and average absolute roll/pitch angular velocity. The success rate is defined as follows. For each target velocity, there is a target traveling distance. If the robot can reach this target traveling distance within 30 seconds, this episode is marked as a success. Both EP and BP-trained policies are tested. For each algorithm, we test three different target velocities, and each target velocity is tested on uneven terrain with six different $\mathbf{H_{max}}$, which indicates the hardness of the uneven terrain. A statistic over 500 episodes of walking test is conducted for each combination of conditions for each training version (note that there are 5 training versions):

algorithm, target velocity, and $\mathbf{H_{max}}$. For the details about the setup of the walking test, see Appendix D. The final statistics is the mean (over 5 versions) of the mean (over 500 episodes) and the standard deviation (over 5 versions) of the mean (over 500 episodes). Note that the means (over 500 episodes) of the six indicators, average forward velocity, average actuator total power, average absolute roll/pitch angle, and average absolute roll/pitch angular velocity, are weighted means. Fig. 3 shows the over-version mean (center mark) and the over-version standard deviation (half length of the error bar) of the seven performance indicators under different testing conditions.

The EP-trained controller achieves broadly comparable success rate, forward velocity, actuator power, and roll/pitch angular-velocity stability to the BP-trained reference in the reported tests. This provides task-level evidence that EP-based training can produce an effective locomotion controller. Evaluation uses fixed policies on random generation of the same box-terrain family used for training, and it does not involve online policy updates. The actuator-power metric describes simulated mechanical behavior, rather than the power consumption of the learning computation (as computational energy is heavily dependent on hardware architectural details and beyond the scope of this paper). Additional evaluation details and statistics are provided in Appendix D.

4.2.1 GPU MEMORY CONSUMPTION

The gradient of the convergent RNN can also be estimated by using global BPTT Werbos (1990). BPTT-based gradient estimation is typically used as a baseline in EP algorithms research to quantify computational benefit Laborieux et al. (2021). We measured the GPU memory consumption of BPTT-based and EP local learning rule-based gradient estimation. In the experiment, the training mini-batch size is 32,768. During stage-1 training, BPTT algorithm consumes 13,486 MB of GPU memory, while EP algorithm consumes 3,110 MB for the same task. Thus, it can be concluded that EP is approximately $4.3\times$ more memory efficient compared with BPTT due to local computations.

4.3 ABLATION STUDIES

The ablations in Appendix E characterize the choices used to obtain successful EP-based locomotion training. The tested reverse-clipping coefficients $\epsilon_{\text{rev}} = 0.3$ and 0.7 support learning, whereas $\epsilon_{\text{rev}} = 1.0$ exhibits instability. The $1/\sigma_i$ scaling reduces the instability observed with $1/\sigma_i^2$ in the reported comparison. The tested static-mask formulation fails to improve the policy, and the iDCT preprocessing changes the observed relaxation-convergence behavior. These results provide empirical evidence about the working training configuration and its failure modes.

5 DISCUSSION AND LIMITATIONS

The main finding of this work is that EP-based local learning can train a quadruped locomotion controller. Comparable performance to the BP-PPO reference is evidence that the local-learning approach produces useful locomotion behavior. The broader motivation is to enable robots to continue learning after deployment without repeatedly returning to an external simulation-and-retraining pipeline. Local EP updates are relevant to this objective because they create opportunities to colocate weight storage, neural computation, and updating in neuromorphic or in-memory hardware Scellier & Bengio (2017); Scellier et al. (2023). Such hardware could reduce the storage and data-movement costs associated with global gradient propagation, making local learning a promising direction for energy-constrained onboard updates. The present study establishes the prerequisite that this hardware-motivated learning mechanism can train the policy and value networks for a quadruped locomotion task. The reported GPU memory reduction relative to BPTT is a supporting computational result. Realized energy and latency benefits will depend on the eventual hardware, peripheral circuitry, and relaxation overhead and are not measured here.

On-robot adaptation remains a central objective of this research direction, but its implementation and evaluation are beyond the scope of the current paper. Extending the proposed framework to physical-robot adaptation will require sample-efficient and safe data collection, implementation of the local neural updates on suitable hardware, and onboard support for the surrounding RL operations, including rollout storage and advantage estimation.

AI USE STATEMENT

In this work, we used generative AI tools to create multi-threading simulation programs, search literature, draft paragraphs, perform grammar checks, perform symbol consistency checks, and edit Python scripts for plotting. We have not used generative AI tools to create any theoretical frameworks, mathematical derivations, or generate the bibliography. We manually checked all AI-generated paragraphs and sections. All the AI-generated code is checked and analyzed by the author. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

For the purpose of reproducing the results shown in this work, the codebase of the locomotion training algorithm is provided in the supplementary material.

REFERENCES

Guillaume Bellegarda and Auke Ijspeert. Cpg-rl: Learning central pattern generators for quadruped locomotion. *IEEE Robotics and Automation Letters*, 7(4):12547–12554, 2022.

Guillaume Bellegarda, Milad Shafiee, and Auke Ijspeert. Visual cpg-rl: Learning central pattern generators for visually-guided quadruped locomotion. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1420–1427. IEEE, 2024.

Michael Bloesch, Jan Humplik, Viorica Patraucean, Roland Hafner, Tuomas Haarnoja, Arunkumar Byravan, Noah Yamamoto Siegel, Saran Tunyasuvunakool, Federico Casarini, Nathan Batchelor, Francesco Romano, Stefano Saliceti, Martin Riedmiller, S. M. Ali Eslami, and Nicolas Heess. Towards real robot learning in the wild: A case study in bipedal locomotion. In Aleksandra Faust, David Hsu, and Gerhard Neumann (eds.), *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1502–1511. PMLR, November 2022.

Sten Grillner and Abdeljabbar El Manira. Current Principles of Motor Control, with Special Reference to Vertebrate Locomotion. *Physiological Reviews*, 100(1):271–320, January 2020. ISSN 0031-9333, 1522-1210. doi: 10.1152/physrev.00015.2019.

Sten Grillner and Alexander Kozlov. The CPGs for Limbed Locomotion–Facts and Fiction. *International Journal of Molecular Sciences*, 22(11):5882, May 2021. ISSN 1422-0067. doi: 10.3390/ijms22115882.

Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 2020.

Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, 9(88):eadi7566, 2024.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural networks*, 21(4):642–653, 2008.

Ole Kiehn. LOCOMOTOR CIRCUITS IN THE MAMMALIAN SPINAL CORD. *Annual Review of Neuroscience*, 29(1):279–306, July 2006. ISSN 0147-006X, 1545-4126. doi: 10.1146/annurev.neuro.29.051605.112910.

Ole Kiehn. Decoding the organization of spinal circuits that control locomotion. *Nature Reviews Neuroscience*, 17(4):224–238, April 2016. ISSN 1471-003X, 1471-0048. doi: 10.1038/nrn.2016.9.

Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. *arXiv preprint arXiv:1909.06586*, 2019.

Hiroshi Kimura, Yasuhiro Fukuoka, and Avis H Cohen. Biologically inspired adaptive walking of a quadruped robot. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1850):153–170, 2007.

Yoshimasa Kubo, Eric Chalmers, and Artur Luczak. Combining backpropagation with Equilibrium Propagation to improve an Actor-Critic reinforcement learning framework. *Frontiers in Computational Neuroscience*, 16:980613, August 2022. ISSN 1662-5188. doi: 10.3389/fncom.2022.980613.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

Axel Laborieux and Friedemann Zenke. Holomorphic equilibrium propagation computes exact gradients through finite size oscillations. *Advances in neural information processing systems*, 35: 12950–12963, 2022.

Axel Laborieux and Friedemann Zenke. Improving equilibrium propagation without weight symmetry through jacobian homeostasis. *arXiv preprint arXiv:2309.02214*, 2023.

Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. *Frontiers in neuroscience*, 15:633674, 2021.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference Target Propagation. In Annalisa Appice, Pedro Pereira Rodrigues, Vítor Santos Costa, Carlos Soares, João Gama, and Alípio Jorge (eds.), *Machine Learning and Knowledge Discovery in Databases*, volume 9284, pp. 498–515. Springer International Publishing, Cham, 2015. ISBN 978-3-319-23527-1 978-3-319-23528-8. doi: 10.1007/978-3-319-23528-8_31.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, October 2020. ISSN 2470-9476. doi: 10.1126/scirobotics.abc5986.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7 (1):13276, November 2016. ISSN 2041-1723. doi: 10.1038/ncomms13276.

Poramate Manoonpong, Ulrich Parlitz, and Florentin Wörgötter. Neural control and adaptive neural forward models for insect-like, energy-efficient, and adaptable locomotion of walking machines. *Frontiers in neural circuits*, 7:12, 2013.

Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *The International Journal of Robotics Research*, 43(4):572–587, 2024.

Erwann Martin, Maxence Ernoult, Jérémie Laydevant, Shuai Li, Damien Querlioz, Teodora Petrisor, and Julie Grollier. Eqspike: spike-driven equilibrium propagation for neuromorphic implementations. *Iscience*, 24(3), 2021.

Wenjuan Ouyang, Haozhen Chi, Jiangnan Pang, Wenyu Liang, and Qinyuan Ren. Adaptive locomotion control of a hexapod robot via bio-inspired learning. *Frontiers in Neurorobotics*, 15:627157, 2021.

Ludovic Righetti and Auke Jan Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *2008 IEEE International Conference on Robotics and Automation*, pp. 819–824. IEEE, 2008.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Benjamin Scellier, Maxence Ernoult, Jack Kendall, and Suhas Kumar. Energy-based learning algorithms for analog computing: a comparative study. *Advances in neural information processing systems*, 36:52705–52731, 2023.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017.

Ryosei Seto, Guanda Li, Kyo Kutsuzawa, Dai Owaki, and Mitsuhiro Hayashibe. Two-stage learning of cpg and postural reflex towards quadruped locomotion on uneven terrain with simple reward. *IEEE Access*, 2025.

Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022.

Shura Suzuki, Kosuke Matayoshi, Mitsuhiro Hayashibe, and Dai Owaki. Foot trajectory as a key factor for diverse gait patterns in quadruped robot locomotion. *Scientific Reports*, 15(1):1861, January 2025. ISSN 2045-2322. doi: 10.1038/s41598-024-84060-5.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

P.J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, October 1990. ISSN 00189219. doi: 10.1109/5.58337.

James C. R. Whittington and Rafal Bogacz. An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. *Neural Computation*, 29(5):1229–1262, May 2017. ISSN 0899-7667, 1530-888X. doi: 10.1162/NECO_a_00949.

APPENDIX

A CONTROL ARCHITECTURE

This appendix provides the implementation details of the CPG–RES controller summarized in Section 3.1, including the oscillator dynamics, foot trajectory generator, joint inverse-kinematics, and residual angle integration. This control architecture follows the two-stage learning of CPG and postural reflex proposed in Seto et al. (2025).

A.1 A1 ROBOT MECHANICAL PROPERTIES

The robot used in this work, Unitree A1, contains 4 limbs, and each limb contains 3 joints: hip, thigh, and calf. The nominal whole body weight used in the MuJoCo simulation is $12.454\,\mathrm{kg}$, provided by the robot manufacturer.

A.2 CPG OSCILLATOR MODEL AND FOOT TRAJECTORY GENERATOR

The Hopf oscillator is used as the core of the CPG following Seto et al. (2025). Four independent Hopf oscillators generate periodic signals for the four feet. The dynamics of the Hopf oscillator for foot i are, $\forall i$

$$\ddot{r}_i = a(\frac{a}{4}(\mu_i - r_i) - \dot{r}_i) \tag{A-1}$$

$$\dot{\theta}_i = \omega_i \tag{A-2}$$

$$\dot{\phi}_i = \psi_i \tag{A-3}$$

Here, μ_i, ω_i, ψ_i are the CPG oscillator parameters that are provided by the CPG policy network at each RL control step. $a = 150\,\mathrm{s}^{-1}$. The r_i, θ_i, ϕ_i are the CPG oscillator states. $r_i \in \mathbb{R}$. θ_i and ϕ_i are wrapped into $[-\pi, \pi]$. r_i, θ_i, ϕ_i directly determine the CPG Cartesian coordinates of the foot i in the robot local frame in the following way, $\forall i$

$$x_{i,\mathrm{CPG}} = -d_{\mathrm{step}}(r_i - 1)\cos(\theta_i)\cos(\phi_i) \tag{A-4}$$

$$y_{i,\mathrm{CPG}} = -d_{\mathrm{step}}(r_i - 1)\cos(\theta_i)\sin(\phi_i) \tag{A-5}$$

$$z_{i,\mathrm{CPG}} = -h + (g_c \textbf{ if } \sin(\theta_i) > 0 \textbf{ else } g_p)\sin(\theta_i) \tag{A-6}$$

Here, $(x_{i,\mathrm{CPG}}, y_{i,\mathrm{CPG}}, z_{i,\mathrm{CPG}})$ are the CPG Cartesian coordinates for foot i. $d_{\mathrm{step}}, h, g_c, g_p$ are the stride length, robot body height, swing clearance, and stance penetration of the foot trajectory, respectively. The d_{step} parameter is fixed at $0.15\,\mathrm{m}$, and h, g_c, g_p are randomized in the training process. For details about the randomized trajectory, see the domain randomization section. Readers are directed to Seto et al. (2025) for more details about the foot trajectory generator. Another work comprehensively discussed the emergence of different locomotion gaits that are determined by the shape of the foot trajectory Suzuki et al. (2025).

The hip, thigh, and calf CPG joint angles of limb i: $(q_{\mathrm{hip},i,\mathrm{CPG}}, q_{\mathrm{thigh},i,\mathrm{CPG}}, q_{\mathrm{calf},i,\mathrm{CPG}})$ are determined by the robot limb inverse kinematics given $(x_{i,\mathrm{CPG}}, y_{i,\mathrm{CPG}}, z_{i,\mathrm{CPG}})$:

$$(q_{\mathrm{hip},i,\mathrm{CPG}}, q_{\mathrm{thigh},i,\mathrm{CPG}}, q_{\mathrm{calf},i,\mathrm{CPG}}) = \mathbf{InverseKinematics}(x_{i,\mathrm{CPG}}, y_{i,\mathrm{CPG}}, z_{i,\mathrm{CPG}}) \quad \forall i \tag{A-7}$$

The RES policy outputs a residual angle derivative for each actuated joint. The residual angle derivative is integrated into the joint residual angle:

$$q_{j,i,\mathrm{RES}}(t) = \mathbf{Clamp}(q_{j,i,\mathrm{RES}}(t-1) + \dot{q}_{j,i,\mathrm{RES}}(t) \cdot \Delta t_{\mathrm{low}}, -\mathbf{ResLimit}, \mathbf{ResLimit}) \quad \forall j, i \tag{A-8}$$

Here $q_{j,i,\mathrm{RES}}(t), \dot{q}_{j,i,\mathrm{RES}}(t)$ are the residual angle and residual angle derivative for the joint j in limb i at time step t. Δt_{low} is the low-level control timestep and will be discussed later. $\mathbf{ResLimit}$ is the bound for residual angle output. Finally, the target joint angle is the sum of the CPG joint angle and the residual angle:

$$q_{j,i,\mathrm{target}} = q_{j,i,\mathrm{CPG}} + q_{j,i,\mathrm{RES}} \quad \forall j, i \tag{A-9}$$

Table 1: Observation spaces of CPG policy network and RES policy network.

Observation	Number of dim. CPG	RES
Actuator angle	12	12
Actuator angular velocity	12	12
IMU (euler)	2	2
IMU (angular velocity)	3	3
IMU (linear acceleration)	3	3
Foot contact	4	4
CPG state	24	24
Command	3	3
Residual angle	–	12
Residual angle derivative	–	12
Total	63	87

Table 2: Action spaces of CPG policy network and RES policy network. Limb index $i \in \{1, 2, 3, 4\}$.

	Action	Range
CPG	μ_i	[1.0, 2.0]
	ω_i	[0.0, 3.0]
	ψ_i	[-1.5, 1.5]
RES	$\dot{q}_{\mathrm{hip},i,\mathrm{RES}}$	[-5.0, 5.0]
	$\dot{q}_{\mathrm{thigh},i,\mathrm{RES}}$	[-5.0, 5.0]
	$\dot{q}_{\mathrm{calf},i,\mathrm{RES}}$	[-5.0, 5.0]

The PD controller has the following form:

$$\tau_{j,i} = \mathbf{Clamp}(-K_p(q_{j,i} - q_{j,i,\mathrm{target}}) - K_d\,\dot{q}_{j,i}, -\mathbf{TorqueLimit}, \mathbf{TorqueLimit}) \quad \forall j, i \tag{A-10}$$

Here, $\tau_{j,i}$, $q_{j,i}$, $q_{j,i,\mathrm{target}}$, and $\dot{q}_{j,i}$ are joint torque to be applied, current joint angle, target joint angle, current angular velocity of joint j at limb i, respectively. Two PD hyper-parameters are $K_p = 100\,\mathrm{N\,m}$ and $K_d = 2\,\mathrm{N\,m\,s}$. Also, all the calculated joint torques are clamped to the range of $[-\mathbf{TorqueLimit}, \mathbf{TorqueLimit}]$ before being applied to the MuJoCo environment.

The RL controller (CPG and RES policy networks) inference frequency is 100 Hz, i.e., $\Delta t_{\mathrm{RL}} = 0.01\mathrm{s}$. The low-level controllers, including the PD controller, CPG oscillator dynamics, and residual angle dynamics, operate at 1000 Hz. i.e., $\Delta t_{\mathrm{low}} = 0.001\mathrm{s}$. It implies that for each pair of outputs from the CPG policy network and the RES policy network, the CPG oscillator and residual angle dynamics step forward by 10 times using the same input parameters.

A.3 Observation and action spaces

The definition of the dimensions in the observation space and action space is shown in Table 1 and Table 2, which are identical to the observation and action spaces in Seto et al. (2025). Note that the binary foot contact dimension is 1 when the contact force between the foot and the terrain is greater than 0.1 N and 0 otherwise. Also, the user command contains 3 dimensions: local x-directional target velocity, local y-directional target velocity, and yaw target angular velocity. The "local" here refers to the local coordinate frame fixed on the robot. In the local frame, positive x-direction is the heading forward direction, positive y-direction points to the left side, and positive z-direction is the vertically upward direction. In the training process, the local y-directional target velocity and yaw target angular velocity are constantly zero in this locomotion task, i.e., this locomotion task does not require any lateral velocity or turning of the robot. For the local x-directional target velocity, at the beginning of each episode, a $v_{x,\mathrm{local,target}}$ will be sampled from $\mathbf{Uniform}(0, v_{x,\mathrm{upper}})$ where $v_{x,\mathrm{upper}}$ is the upper bound of the local x-directional target velocity.

A.4 Reward function

The reward function indicates the accuracy of the velocity tracking, the intensity of the robot vibration, and the locomotion energy consumption. Based on the simple reward function proposed in Seto et al. (2025); Bellegarda & Ijspeert (2022), modifications are made in the x-directional linear velocity tracking term and its coefficient. The details about the reward function are shown in Table 3. The closeness functions $f_{r1}(\cdot)$ and $f_{r2}(\cdot)$ are defined as:

$$f_{r1}(x) = \exp\left(-\frac{x^2}{d_{r1}}\right), \quad f_{r2}(x) = \exp\left(-\frac{x^2}{0.25}\right) \tag{A-11}$$

Here, d_{r1} is 0.25 in stage-1 and 0.04 in stage 2.

Table 3: Reward Function. The terms in the RL reward function. The reward coefficient for the x-directional linear velocity tracking is different in the stages. The details are in the Reward function section.

Reward term	**Coefficient**	**Description**
$f_{r1}(v_{x,\text{local}} - v_{x,\text{local,target}})$	$\alpha_{vx}\Delta t_{\text{RL}}$	x-directional linear velocity tracking
$f_{r2}(v_{y,\text{local}} - v_{y,\text{local,target}})$	$0.75\Delta t_{\text{RL}}$	y-directional linear velocity tracking
$f_{r2}(\omega_{\text{yaw}} - \omega_{\text{yaw,target}})$	$0.50\Delta t_{\text{RL}}$	yaw angular velocity tracking
$-v_{z,\text{local}}^2$	$2.00\Delta t_{\text{RL}}$	z-directional linear velocity velocity penalty
$-(\omega_{\text{roll}}^2 + \omega_{\text{pitch}}^2)$	$0.05\Delta t_{\text{RL}}$	roll and pitch angular velocity penalty
$-\sum_{i=1}^{D_{\text{action}}} \tau_i \dot{q}_i$	$0.001\Delta t_{\text{RL}}$	total motor power

Similarly, the reward coefficient for the x-directional linear velocity tracking α_{vx} is different for the two stages. α_{vx} is 3.00 in stage-1 and 6.00 in stage-2.

Note that the linear velocities are the velocities of the torso in the robot frame. The $v_{x,\text{local,target}}, v_{y,\text{local,target}}, \omega_{\text{yaw,target}}$ are the 3-D user commands, and they are included in the observation vector of both stage-1 and stage-2.

A.5 Episode setup and training paradigm

For episodic training, the maximum episode length is 2000 RL control steps, which is equivalent to $20\,\text{s}$. The episode will end early if the torso or any thigh of the robot contacts the terrain.

The entire training paradigm can be split into two stages: stage-1 and stage-2. In stage-1, the CPG policy network is trained alone on flat ground, so that it can maintain a robust forward walking gait and to track the target velocity command. The RES policy network is not enabled in stage-1. The CPG joint angles output from the foot trajectory generators are directly converted into the joint torques without adding any residual angle. In stage-2, the CPG policy network is fixed and does inference only, and the RES policy network is trained. Each episode in stage-2 uses randomly generated uneven terrain. The trained RES policy network is expected to adjust the output from the foot trajectory generator by adding residual angles onto the CPG target joint angles. The RES policy maintains the robot's stability and overcomes the terrain obstacles. Domain randomization is used in both stage-1 and stage-2, but the randomness in these two stages is slightly different.

B SIMULATION ENVIRONMENT AND EPISODE INITIALIZATION

MuJoCo platform Todorov et al. (2012) is used to simulate the locomotion task. MuJoCo is configured so that its forward time step size is the same as the low-level controller time step size. i.e., $\Delta t_{\mathrm{MuJoCo}} = \Delta t_{\mathrm{low}}$. From a programming perspective, every low-level controller step is followed by a MuJoCo forward step.

B.1 DOMAIN RANDOMIZATION

The details of domain randomization for both stages are shown in Table 4. The majority of the randomized hyper-parameters align with the ones in Seto et al. (2025). However, the frequency of random push is increased. Note that the limb mass variation ratio is multiplied by every single limb link mass at the environment resetting. i.e., for each link (Unitree A1 has a total of 12 limb links), a new random mass ratio is generated and multiplied by the link mass. Also, the random push is applied horizontally (i.e., parallel with the x-y plane in global frame) with magnitude $0.5\,\mathrm{m/s}$. When implementing in MuJoCo, the random push is realized by applying an external force F_{push} to the robot torso for 10 Δt_{low}. F_{push} satisfies

$$|F_{\mathrm{push}}| \cdot 10 \cdot \Delta t_{\mathrm{low}} = M_{\mathrm{robot}} \cdot 0.5\,\mathrm{m/s}$$

Here, $|F_{\mathrm{push}}|$ is the magnitude of the pushing force, and M_{robot} is the mass of the entire robot in the current episode. The sequence of the timing of random push is a Poisson train, which is approximated through sampling a Bernoulli random variable in every RL control step.

Table 4: Domain randomization in training

Environment hyper-parameter	**Range/Value**
MuJoCo ground friction sliding coefficient	stage-1,2: $[0.5, 2.5]$
Limb mass ratio	stage-1,2: $[0.5, 1.5]$
Torso load (kg)	stage-1,2: $[0.0, 5.0]$
Robot body height h (m)	stage-1,2: $[0.22, 0.32]$
Swing clearance g_c (m)	stage-1: $[0.03, 0.20]$, stage-2: $[0.15, 0.20]$
Stance penetration g_p (m)	stage-1,2: $[0.00, 0.02]$
Random push average interval (s)	stage-1,2: 5

The domain randomized environment and foot trajectory hyper-parameters are fixed in the evaluation, which is shown in Table 5.

Table 5: Fixed walking testing hyper-parameters

Environment hyper-parameter	**Value**
MuJoCo ground friction sliding coefficient	1.5
Limb mass ratio	1.0
Torso load (kg)	0.0
Terrain box side length (m)	0.4
Robot body height h (m)	0.25
Swing clearance g_c (m)	0.1
Stance penetration g_p (m)	0.02
Random push	Disabled

B.2 UNEVEN TERRAIN GENERATION

In stage-2, when resetting the environment at the beginning of each episode, a new uneven terrain is generated by adding geom boxes onto the flat ground. All the boxes are rectangular prisms with square bases. The shape of a box has two parameters: the **side length** and the **height**. Before the beginning of each episode, a single side length is sampled from a distribution $\mathbf{Uniform}(\mathbf{SideLength_{low}}, \mathbf{SideLength_{high}})$, and this side length is applied to all

the terrain boxes for this episode. The height of each box is sampled from the distribution $\mathbf{Uniform}(10^{-4}\mathrm{m}, \mathbf{H_{max}})$. As all the boxes have the same base side length, they are horizontally densely arranged on the ground without any interval. The maximum box height $\mathbf{H_{max}}$ determines the difficulty of the terrain. In stage-2 training, $\mathbf{H_{max}}$ is set to $0.12\,\mathrm{m}$ constantly for all episodes. In the locomotion performance evaluation, the success rate, average velocity, and other stability indicators are measured at different $\mathbf{H_{max}}$. Details about the locomotion performance evaluation are in the Experiments section.

B.3 Robot state initialization

At the beginning of each episode, we initialize the CPG states, residual angles, robot pose, and user commands as follows. The CPG oscillators are initialized in a random manner. First, a random number θ_a is sampled from distribution $\mathbf{Uniform}(-\pi, \pi)$. Then $\theta_b = (\theta_a + \pi \textbf{ if } \theta_a \leq 0 \textbf{ else } \theta_a - \pi)$ is created, i.e., θ_b is shifted from θ_a by π, and θ_b is wrapped into $[-\pi, \pi]$. For all the limbs, initial r_i is sampled from $\mathbf{Uniform}(1, 2)$ independently. $\dot{r}_i$ is initialized to zero. Initial θ_1 and θ_4 are set to θ_a. Initial θ_2 and θ_3 are set to θ_b. Initial ϕ_i is sampled from $\mathbf{Uniform}(-\frac{\pi}{12}, \frac{\pi}{12})$ independently. The residual angle values are initialized as zeros. The joint angles in MuJoCo state vector are directly set to the target joint angles.

At initialization, a free-falling initialization is used to create a natural and valid initial pose of the robot in the global world. i.e., the robot is released from a fixed location above the ground, and the robot will do a free-fall until at least one of the feet touches the ground. During the free-fall, the overall target joint positions are fixed to the initial values, and the PD controller runs at each MoJoCo timestep to maintain roughly fixed joint positions. No RL actor inference or training happens during the free-fall. When the contact between the feet and the ground is detected, the velocities of all the DoFs of the robot are set to zero. Therefore, the robot is now frozen so that it has an upright posture with zero velocity and is touching the ground. After one more MuJoCo forward step, the robot is well-prepared to start the first RL control step.

The spatial reference point of the robot is located inside the trunk, which is set up by the robot manufacturer and defined within the MuJoCo model specifications. Before the free-fall, the spatial reference point of the robot is set to $(0, 0, 0.5)\,\mathrm{m}$ in the global frame of the MuJoCo world. In this case, the robot is guaranteed to be above the ground and not to contact the ground, neither on flat ground nor on uneven terrain.

C EP-BASED RL TRAINING

The PPO variant adopted in this paper contains three trainable parts: policy network, value network, and the log-std vector. In the work, two EP networks, a type of recurrent neural network with bi-directional connections, instead of standard feed-forward networks, play the role of the CPG policy network and the RES policy network. The log of action standard deviation is a standalone vector and will be updated individually instead of being part of the policy network output. The EP policy only outputs the mean of action.

C.1 PPO TRAINING PROCESS OVERVIEW

The entire PPO training process can be split into a sequence of PPO training units. A PPO training unit contains a rollout phase and a neural network update phase. In the rollout phase, the robot is directed to walk/run in each environment for T control steps. In the case of simulation, the total number of environments is N. At each control step, firstly, an observation vector s_t is generated by the robot. The observation vector contains external information (for example, user command), and internal information (for example, the IMU sensor and foot contact sensor output). According to the observation vector, the policy network outputs an action mean vector μ_t. In an actual deployment and performance testing situation, this action mean vector is directly used for robot control. However, in the training process, some new actions need to be explored for developing potentially better locomotion strategies. Therefore, in the training process, a noise vector ϵ_t will be added to the action mean vector, resulting in the action vector $a_t = \mu_t + \epsilon_t$. The noise vector ϵ_t is usually sampled from a zero-mean normal distribution $\mathcal{N}(0, \sigma)$. Or equivalently, it can be described that the action a_t is sampled from the distribution $\mathcal{N}(\mu_t, \sigma)$. It should be noted that σ is from a standalone log-std vector. At the same time, the logarithm of the probability of sampling this action vector $\log(\pi(a_t|s_t))$ will be recorded. $\pi(a_t|s_t) = f_{\mathcal{N}(\mu_t,\sigma)}(a_t)$, where f is the probability density function of $\mathcal{N}(\mu_t, \sigma)$. In the rollout dataset, this log probability will be marked as $\log(\pi_{\text{rollout}}(a_t|s_t))$ because if the policy network is updated, the log probability of $a_t|s_t$ will change. The "rollout" marker distinguishes it from an after-policy-update log probability or current log probability $\log(\pi(a_t|s_t))$. The action vector is applied in the robot control, and the environment time step is forwarded by Δt_{RL}. Then, the reward function outputs a scalar reward r_t according to the stability, velocity, and energy consumption of the locomotion, and the robot will receive the next step observation vector s_{t+1}. Subsequently, the value function for current and next step observation $\mathbf{Value}(s_t)$, $\mathbf{Value}(s_{t+1})$ are calculated through value network inference. Finally, from the environment, a done signal d_t, which indicates the end of the current episode, and a robot falling signal fall_t, which indicates whether the robot fell in the current step, will be collected. It should be noted here that, if the maximum episode length is reached but the robot has not fallen, future reward is assumed non-zero. In this case, the value function of the next state will be used as the bootstrapped future cumulative reward after the final step of the episode. In summary, at time t, the following data tuple will be stored in the buffer

$$(s_t, a_t, r_t, \log(\pi_{\text{rollout}}(a_t|s_t)), d_t, \text{fall}_t, \mathbf{Value}(s_t), \mathbf{Value}(s_{t+1}))$$

For the simulation case, if there are N parallel environments, a total of $N \cdot T$ data tuples will be collected in the rollout phase.

In the neural network update phase, the data tuples stored in the buffer will be processed. $\forall t$, first, $\mathbf{Return}(s_t)$ and $\mathbf{Advantage}(s_t, a_t)$ are calculated through GAE Schulman et al. (2015), which requires r_t, d_t, fall_t and $\mathbf{Value}(s_t), \mathbf{Value}(s_{t+1})$. Afterwards, the policy network and the value network are updated for K_{epoch} epochs and $\frac{|\mathcal{D}|}{|\mathcal{B}|}$ mini-batches in each epoch, where $\mathcal{D}$ is a single rollout dataset with $N \cdot T$ data tuples, and $\mathcal{B}$ is a data mini-batch. The policy network is updated by maximizing the PPO action surrogate objective function, which will be discussed in a later section. This step requires s_t, a_t, $\log(\pi_{\text{rollout}}(a_t|s_t))$, $\mathbf{Advantage}(s_t, a_t)$ and σ_t. Finally, the value network is updated by minimizing the L_2 loss between the value network output and the return:

$$\frac{1}{|\mathcal{B}|} \sum_t (\mathbf{Value}(s_t) - \mathbf{Return}(s_t))^2 \tag{A-12}$$

C.2 BACKPROPAGATION-BASED POLICY UPDATE

The PPO action surrogate objective is

$$L^{\text{CLIP}} = \hat{\mathbb{E}}_t[\min(r_t(\mu_t)\hat{A}_t, \mathbf{Clamp}(r_t(\mu_t), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{A-13}$$

Here, $\hat{A}_t$ is the estimated $\mathbf{Advantage}(s_t, a_t)$ and ϵ is the PPO ratio clip coefficient. $r_t(\mu_t)$ is the probability ratio between the current action probability $\pi(a_t|s_t)$ and the action probability stored in the rollout $\pi_{\text{rollout}}(a_t|s_t)$, i.e., $r_t(\mu_t) = \frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)}$. Then, backpropagation is used to calculate the objective gradient at μ_t and further propagate the gradient back and update the policy and value network parameters to maximize the PPO surrogate objective and to minimize the value function loss.

C.3 EP-BASED POLICY NETWORK UPDATE

The EP network training contains a free phase and one or more nudge phases Scellier & Bengio (2017); Laborieux et al. (2021). In different phases, the entire network converges to different equilibrium states determined by the external perturbation from the output layer. In this paper, the one-free-two-nudge version of EP is used for optimal gradient estimation Laborieux et al. (2021). For each mini-batch, the policy and value networks individually experience a free phase, a positive nudge phase, and a negative nudge phase.

For the value network, the loss function is still of the form in Equation A-12, which is a simple regression problem. Only the policy network update will be discussed in detail here.

For the policy network, the input is the observation vector or the pre-processed observation vector. A modified PPO objective gradient is applied to the output layer neurons in the nudge phases. To find the gradient of the PPO action surrogate objective, the probability ratio $r_t(\mu_t)$ from the objective (Equation A-13) can be expanded as:

$$r_t(\mu_t) = \frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)} = \exp(\log(\pi(a_t|s_t)) - \log(\pi_{\text{rollout}}(a_t|s_t))) \tag{A-14}$$

Given that the dimension i of action a_t is sampled from an independent normal distribution with mean $\mu_{t,i}$ and standard deviation σ_i, the probability density of getting action a_t given observation s_t is:

$$\pi(a_t|s_t) = \prod_i^{D_{\text{action}}} \left[\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(a_{t,i} - \mu_{t,i})^2}{2\sigma_i^2} \right) \right] \tag{A-15}$$

Here, D_{action} is the dimension of the action space. $\mu_{t,i}$ is the i-th dimension of the output of the policy network given s_t and σ_i is the i-th dimension of the standard deviation vector. It can be derived that the gradient of the probability ratio $r_t(\mu_t)$ w.r.t. $\mu_{t,i}$ is:

$$\begin{aligned} \frac{\partial r_t(\mu_t)}{\partial \mu_{t,i}} &= \frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)} \frac{\partial \log \pi(a_t|s_t)}{\partial \mu_{t,i}} \\ &= \frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)} \frac{(a_{t,i} - \mu_{t,i})}{\sigma_i^2} \end{aligned} \tag{A-16}$$

Note that the probability ratio $\frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)}$ is always close to 1 in the training process. Thus, we can write:

$$\frac{\partial r_t(\mu_t)}{\partial \mu_{t,i}} \approx \frac{(a_{t,i} - \mu_{t,i})}{\sigma_i^2} \tag{A-17}$$

Therefore, if $\mu_{t,i}$ is the i-th dimension of the action mean in sample t, we have the approximate PPO objective gradient w.r.t. the policy network output $\mu_{t,i}$:

$$\begin{aligned} \frac{\partial L^{\text{CLIP}}}{\partial \mu_{t,i}} &= \frac{1}{|\mathcal{B}|} \begin{cases} \mathbf{1}_{[0,1+\epsilon)}(r_t(\mu_t)) \cdot r_t(\mu_t) \frac{(a_{t,i} - \mu_{t,i})}{\sigma_i^2} \hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \mathbf{1}_{(1-\epsilon,\infty)}(r_t(\mu_t)) \cdot r_t(\mu_t) \frac{(a_{t,i} - \mu_{t,i})}{\sigma_i^2} \hat{A}_t & \text{if } \hat{A}_t < 0 \end{cases} \\ &\approx \frac{1}{|\mathcal{B}|} \begin{cases} \mathbf{1}_{[0,1+\epsilon)}(r_t(\mu_t)) \cdot \frac{(a_{t,i} - \mu_{t,i})}{\sigma_i^2} \hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \mathbf{1}_{(1-\epsilon,\infty)}(r_t(\mu_t)) \cdot \frac{(a_{t,i} - \mu_{t,i})}{\sigma_i^2} \hat{A}_t & \text{if } \hat{A}_t < 0 \end{cases} \end{aligned} \tag{A-18}$$

Here, the terms $\mathbf{1}_{[0,1+\epsilon)}(r_t(\mu_t))$ and $\mathbf{1}_{(1-\epsilon,\infty)}(r_t(\mu_t))$ can be regarded as gradient masks. It sets the continuous part of the gradient $\frac{(a_{t,i}-\mu_{t,i})}{\sigma_i^2}\hat{A}_t$ to zero when the probability ratio is beyond the gradient enabling range (positive advantage: $[0, 1+\epsilon)$, negative advantage: $(1-\epsilon, \infty)$).

By using an EP network to optimize the PPO objective, we can apply the PPO objective gradient w.r.t. the network output $\frac{\partial L^{\text{CLIP}}}{\partial \mu_{t,i}}$ to the output layer neurons in the EP network. The $\mu_{t,i}$ is the policy network output dimension i in BP-trained RL, which is equivalent to $\xi^*_{t,out,i}$ in EP. $\xi^*_{t,out,i}$ is the output neuron state i in free phase equilibrium when the policy input observation is s_t. **It should be noted here that the subscripts t in $\xi^*_{t,out,i}$ represent neither EP relaxation step index nor the time in EP neuron state dynamics, instead, it represents the rollout data sample index, like the t in μ_t.** The output neuron state dynamics modulated by the PPO objective gradient is (see main text):

$$\frac{d\xi_{t,out,i}}{d[\text{time}]} = -\xi_{t,out,i} + \rho'(\xi_{t,out,i})\sum_j w_{out,ij}\rho(\xi_{t,j}) + b_{out,i}\rho'(\xi_{t,out,i}) \tag{A-19}$$
$$- \beta \cdot \frac{1}{|\mathcal{B}|}\begin{cases}\mathbf{1}_{[0,1+\epsilon)}(r_{t,\text{nudging}}(\xi_{t,out}))\cdot\frac{(a_{t,i}-\xi_{t,out,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t \geq 0\\ \mathbf{1}_{(1-\epsilon,\infty)}(r_{t,\text{nudging}}(\xi_{t,out}))\cdot\frac{(a_{t,i}-\xi_{t,out,i})}{\sigma_i^2}\hat{A}_t & \text{if } \hat{A}_t < 0\end{cases}$$

However, based on experiments, this EP output neuron dynamics (Equation A-19) leads to failure in training convergence. The per-update KL-divergence graph indicates excessively large update steps. We investigated the cause of the large update steps. In conventional PPO, backpropagation directly calculates the parameter gradient. But the nudge phases of EP move the output layer neurons in two-directions: the **closer-to-target direction** or the **farther-away-from-target** direction. For example, according to Equation A-19, when $\beta = \beta_{\text{ep}}$ and $\hat{A}_t > 0$, the output neurons are moving in a direction that increase its distance from the action sample, because the objective gradient is proportional to $\xi_{t,out,i} - a_{t,i}$. More concretely, before the very first positive nudge step

$$-\xi_{t,out,i} + \rho'(\xi_{t,out,i})\sum_j w_{out,ij}\rho(\xi_{t,j}) + b_{out,i}\rho'(\xi_{t,out,i}) = 0 \tag{A-20}$$

Therefore,

$$\frac{d\xi_{t,out,i}}{d[\text{time}]} \propto (\xi_{t,out,i} - a_{t,i}) \quad \text{before first positive nudge phase step} \tag{A-21}$$

If the output neuron state shift along the farther-away-from-target direction is not bounded, $r_{t,\text{nudging}}(\xi_{t,out})$ can be indefinitely close to zero. Similarly, when $\beta = \beta_{\text{ep}}$ and $\hat{A}_t < 0$, there is no upper bound for $r_{t,\text{nudging}}(\xi_{t,out})$, but the normal distribution has a probability density upper bound, which does not allow the value explosion of $r_{t,\text{nudging}}(\xi_{t,out})$. When $\beta = -\beta_{\text{ep}}$ and $\hat{A}_t < 0$, the driving force is also farther-away-from-target. However, ratio lower bound exists in this case: $\mathbf{1}_{(1-\epsilon,\infty)}(r_{t,\text{nudging}}(\xi_{t,out}))$. In backpropagation, there is not an equivalent concept of this reverse driving force. Therefore this is an **EP-only phenomenon**.

To constrain these excessively large shifts, we propose the two-sided PPO ratio clip objective gradient:

$$\frac{\partial L^{\text{TwoSidedCLIP}}}{\partial \xi_{t,out,i}} = \frac{1}{|\mathcal{B}|}\begin{cases}\mathbf{1}_{(1-\epsilon_{\text{rev}},1+\epsilon)}(r_{t,\text{nudging}}(\xi_{t,out}))\cdot\frac{(a_{t,i}-\xi_{t,out,i})}{\sigma_i}\hat{A}_t & \text{if } \hat{A}_t \geq 0\\ \mathbf{1}_{(1-\epsilon,1+\epsilon_{\text{rev}})}(r_{t,\text{nudging}}(\xi_{t,out}))\cdot\frac{(a_{t,i}-\xi_{t,out,i})}{\sigma_i}\hat{A}_t & \text{if } \hat{A}_t < 0\end{cases} \tag{A-22}$$

Here, ϵ_{rev} is the reverse PPO ratio clip coefficient. The non-zero gradient is applied to the output layer only when the output neuron state is within the range of the two-sided bounded gradient enabling range.

Additionally, the exploration noise standard deviation σ_i, in the conventional $[-1, 1]$ action space, is usually smaller than 1 and can potentially result in a very large magnitude of the gradient due to the inverse quadratic variation, since the equilibrium shift associated with EP learning is very sensitive to the gradient magnitude.

Our experiments show that this two-sided bounded gradient and reduced gradient scaling factor $\frac{1}{\sigma_i}$ successfully leads to training convergence. The ablation studies involving different values of ϵ_{rev} and $\frac{1}{\sigma_i}$ are shown in Appendix E.

C.4 LOG-STD VECTOR UPDATE

To prevent using backpropagation in the entire algorithm, the gradient of the PPO objective and entropy loss w.r.t. the log-std vector is also explicitly calculated. It needs to be noted that the log-std vector here has the same dimensionality as the action space, which is 12. Note that the original PPO objective is used. The PPO objective gradient w.r.t. the i-th dimension of the log-std vector is

$$\frac{\partial L^{\text{CLIP}}}{\partial \log(\sigma_i)} = \frac{1}{|\mathcal{B}|} \sum_t \begin{cases} \mathbf{1}_{[0,1+\epsilon)}(r_t(\sigma)) \cdot \left[\frac{(a_{t,i}-\mu_{t,i})^2}{\sigma_i^2} - 1 \right] r_t(\sigma) \hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \mathbf{1}_{(1-\epsilon,\infty)}(r_t(\sigma)) \cdot \left[\frac{(a_{t,i}-\mu_{t,i})^2}{\sigma_i^2} - 1 \right] r_t(\sigma) \hat{A}_t & \text{if } \hat{A}_t < 0 \end{cases} \tag{A-23}$$

Here, the probability ratio r_t is regarded as a function of σ, as we only consider the effect of the shift of σ on the shift of r_t.

$$r_t(\sigma) = \frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)} = \exp(\log(\pi(a_t|s_t)) - \log(\pi_{\text{rollout}}(a_t|s_t))) \tag{A-24}$$

The entropy loss w.r.t. the i-th dimension of the log-std vector is

$$\frac{\partial L_{\text{entropy}}}{\partial \log(\sigma_i)} = 2 \cdot k_{\text{entropy}} \cdot (H(\log(\sigma)) - H_{\text{target}}) \tag{A-25}$$

Here, k_{entropy} is the entropy loss coefficient, and H_{target} is the target entropy determined by the entropy scheduler. The entropy of a normal distribution with an uncorrelated per-dimension-log-std vector is:

$$H(\log(\sigma)) = \frac{1}{2} \sum_{i=1}^{D_{\text{action}}} (\log(2\pi e) + 2\log(\sigma_i)) \tag{A-26}$$

The total gradient applied to the i-th dimension of log-std vector is

$$\frac{\partial L^{\text{CLIP}}}{\partial \log(\sigma_i)} + \frac{\partial L_{\text{entropy}}}{\partial \log(\sigma_i)} \tag{A-27}$$

In fact, the entropy loss gradient w.r.t. each dimension of the log-std vector is identical.

C.5 EP NETWORK ARCHITECTURE

Both policy and value networks receive a 1024-D iDCT converted observation vector. Both networks have 2 hidden layers and 768 neurons in each hidden layer. The policy network outputs a 12-D action mean, and the value network outputs a 1-D value function. The EP networks use bias for each neuron in the hidden and output layers. Neuron activation function applied to the hidden layer neurons is hard-sigmoid ($\mathbf{Clamp}(\cdot, \mathbf{0}, \mathbf{1})$). No activation function is applied to the input and output layer neurons. The weight matrix in the EP network is initialized through sampling each weight element independently from the following distribution:

$$\mathbf{Uniform}\left(-\frac{\alpha_w}{\sqrt{[\#\text{Fan_left}]}}, \frac{\alpha_w}{\sqrt{[\#\text{Fan_left}]}} \right) \tag{A-28}$$

Here, α_w is the EP weight initialization scale, and $[\#\text{Fan_left}]$ represents the number of neurons in left layer. From the perspective of a weight matrix, the left layer represents the neuron layer that is closer to the input layer.

D DETAILS ABOUT WALKING TEST AND EXTENDED EXPERIMENTS

D.1 WALKING TEST SETUP

In the 5-meter walking test proposed in Seto et al. (2025), the robot is expected to walk for 5 meters within 30 seconds with $v_{x,\text{local,target}} = 0.3\,\text{m/s}$. The distance is calculated by integrating $v_{x,\text{local}}$ across time. If the robot successfully reaches 5 meters within 30 seconds without falling, this trial is marked as a success. In cases where the robot falls or cannot reach 5 meters within 30 seconds, this trial is marked as a failure. The test is repeated under the condition of 7 different $\mathbf{H_{max}}$ in the set $\{0, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12\}\text{m}$.

In this paper, we used an extended version of the 5-meter walking test by testing the locomotion policy with three different local x-directional target velocity $v_{x,\text{local,target}} \in \{0.1, 0.3, 0.5\}\text{m/s}$. For different local x-directional target velocities, the target traveling distance is also proportionally scaled, i.e., when $v_{x,\text{local,target}} = 0.1\,\text{m/s}$, the target traveling distance is $5\,\text{m} \times (0.1\,\text{m/s})/(0.3\,\text{m/s}) = 1.667\,\text{m}$. Similarly, the target traveling distance for $v_{x,\text{local,target}} = 0.5\,\text{m/s}$ is $8.333\,\text{m}$. We also removed $0\,\text{m}$ from the list of $\mathbf{H_{max}}$ as the flat ground performance will always be better than any non-zero $\mathbf{H_{max}}$. All the testing combinations is the following product set:

$$\{\text{EP}, \text{BP}\} \times \{0.1, 0.3, 0.5\}\text{m/s} \times \{0.02, 0.04, 0.06, 0.08, 0.10, 0.12\}\text{m} \tag{A-29}$$

The following locomotion performance indicators are recorded: success rate, average forward velocity, average actuator total power, average absolute roll/pitch angle, and average absolute roll/pitch angular velocity.

Each of the 5 versions is tested individually. For each version, under the condition of each combination, we run 500 episodes of uneven terrain locomotion. It should be noted that the length of each episode could be different. For each version and each combination of (EP/BP, $v_{x,\text{local,target}}$, $\mathbf{H_{max}}$), we calculated the success rate for the 500 episodes. We also calculated the weighted average of other performance indicators. The weight is the time duration of the episode. For example, a 20-second-long episode has a proportionally larger weight than a 5-second-long episode.

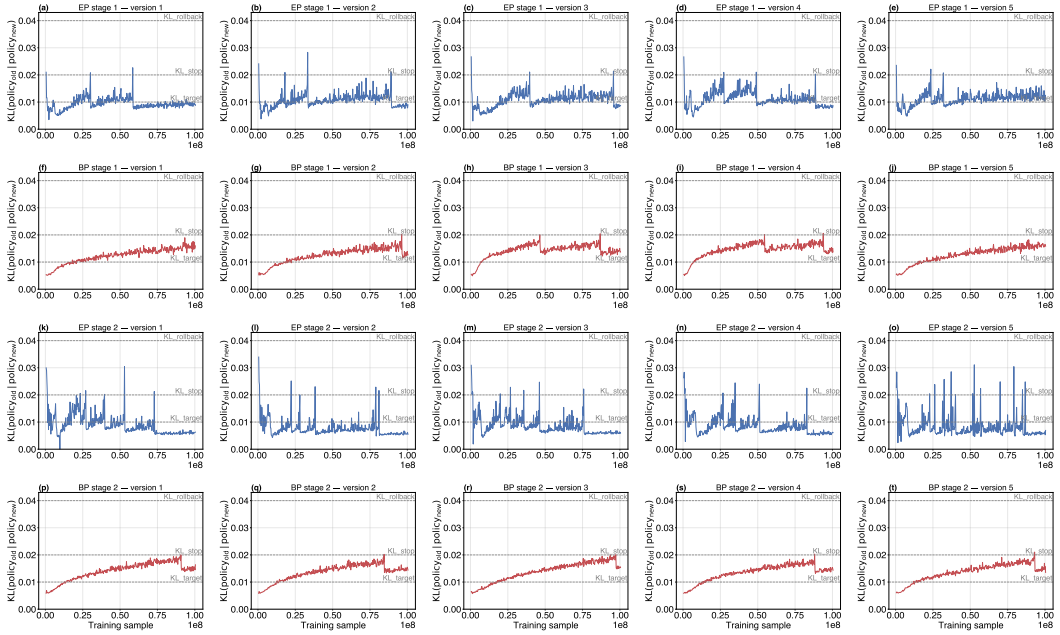

Figure 4: Per-update KL-divergence. The KL-divergence of all 5 versions, EP and BP algorithms, and both stages are plotted here. Here, the y-axis label $\text{KL}(\text{policy}_{\text{old}}|\text{policy}_{\text{new}})$ is the KL-divergence between the pre-update and post-update policy.

D.2 PER-UPDATE KL-DIVERGENCE IN EP AND BP-BASED TRAINING PROCESS

The main paper shows the figure for the rollout mean reward and value MSE in the training process of EP and BP-based locomotion training. In Fig. 4, the per-update KL-divergence (KL-div) for all 5 versions of stage-1 and stage-2 training is plotted. It is clear that BP yields a smoother increment of KL-div in the training process, while EP-based training hits the early stop KL threshold multiple times. The hypothetical cause is that the EP policy network has a non-linear response to the objective gradient, and a small change in the objective gradient leads to a large difference in the KL-div.

D.3 EP NUDGE PHASE DYNAMICS WITH PPO OBJECTIVE GRADIENT

Given an observation vector s_t, the policy network returns an action mean vector μ_t. The intuitive idea of PPO is that, if, for observation s_t, there is any action a_t satisfying $\mathbf{Advantage}(s_t, a_t) > 0$, make μ_t be closer to a_t. Conversely, if $\mathbf{Advantage}(s_t, a_t) < 0$, make μ_t be farther away from a_t. Also, it is critical to maintain small update steps.

This leads us to the question: if we update the policy network according to $\mathbf{Advantage}(s_t, a_t)$ for a group of samples, by how much will corresponding μ_t shift? The intuition is that μ_t will shift significantly if $\mathbf{Advantage}(s_t, a_t)$ has larger magnitude, and μ_t will have minimal shift if $\mathbf{Advantage}(s_t, a_t)$ is close to zero. From the perspective of the PPO, the change of $\frac{\pi(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)}$ should be considered instead of the shift of μ_t. $\pi_{\text{rollout}}(a_t|s_t)$ gives a reference point during policy update: the updated policy should not be very distant from the reference point. The advantage-probability ratio relationship is the core of PPO. Excessively large update step yields policy collapse. For EP-based PPO, investigating the advantage-r_{nudging} relationship is important for maintaining stable policy improvement.

$$
\begin{aligned}
r_{t,\text{nudging}}(\xi_{t,out}) &= \frac{\pi_{\text{nudging}}(a_t|s_t)}{\pi_{\text{rollout}}(a_t|s_t)} \\
&= \frac{1}{\pi_{\text{rollout}}(a_t|s_t)} \prod_{i}^{D_{\text{action}}} \left[\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(a_{t,i} - \xi_{t,out,i})^2}{2\sigma_i^2} \right) \right]
\end{aligned}
\tag{A-30}
$$

According to the definition of $r_{t,\text{nudging}}(\xi_{t,out})$ in Equation A-30, $r_{t,\text{nudging}}(\xi_{t,out})$ will change across relaxation steps in EP network nudge phases. In the positive nudge phase, if the advantage is greater than zero, $r_{t,\text{nudging}}(\xi_{t,out})$ should decrease to a value that is less than 1, i.e., the $\pi_{\text{nudging}}(a_t|s_t)$ is expected to be smaller than the $\pi_{\text{rollout}}(a_t|s_t)$.

To further analyze the relationship between the normalized advantage and $r_{t,\text{nudging}}(\xi_{t,out})$, we recorded the policy output-layer state ξ_{out} during the nudge phases. The $r_{t,\text{nudging}}(\xi_{t,out})$ is calculated by using the action, rollout probability, and log-std in the data analysis. In a training process, after a specific rollout data collection, two separate testing nudge phases were conducted. We randomly chose 1000 samples from the rollout dataset. In these two nudge phases, the value of $\xi_{t,out}$ for each sample t was recorded for each relaxation step. Through this pair of nudge phases, we can know how $\xi_{t,out}$ is changing in the nudge phases according to its advantages. As these 1000 samples are randomly selected, their normalized advantage is approximately normally distributed.

Thus, we can pose the following questions: for a sample with specific advantage $\hat{A}_t$, by how much will the corresponding $\xi_{t,out}$ change in the nudge phase? In which direction will $\xi_{t,out}$ change? By how much will $r_{t,\text{nudging}}(\xi_{t,out})$ change with $\xi_{t,out}$? Our experiments show that, although the after-update probability ratio $r_t(\xi_{t,out})$ changes very slightly, in the nudge phases, $r_{t,\text{nudging}}(\xi_{t,out})$ shifts significantly. $r_{t,\text{nudging}}(\xi_{t,out})$ can reach values as low as 10^{-5} or as high as 10^3, although the PPO ratio clip coefficient is 0.2 (which tries to limits the $r_t(\xi_{t,out})$ to a range $[0.8, 1.2]$). We observed that larger the magnitude of the advantage, larger is the change of $r_{t,\text{nudging}}(\xi_{t,out})$. However, $r_{t,\text{nudging}}(\xi_{t,out})$ is always positive because it is a ratio between probability densities. It is easy to visualize the change of $r_{t,\text{nudging}}(\xi_{t,out})$ in a log scale. In Fig. 5, $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ is plotted versus the advantage $\hat{A}_t$. It can be observed that $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ changes approximately proportionally to the advantage with different coefficients in positive and negative phases. Actually, $\xi_{t,out}$ and $r_{t,\text{nudging}}(\xi_{t,out})$ keep oscillating in the nudging phase. This phenomenon is consistent with the nudge driving force in Equation A-22. To understand the characteristics of this advantage-r_{nudging} relationship more clearly, we collected more nudge phase data. Instead

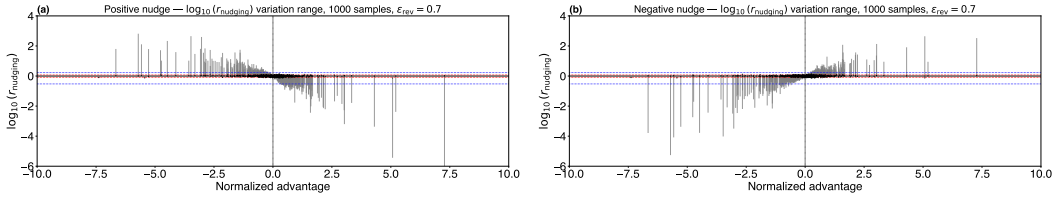

Figure 5: The advantage-r_{nudging} relationship of 1000 samples in the middle of a training process. The variation of $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ in positive and negative nudge phases are shown. The black dots represent the $\log_{10}(r_t(\xi^*_{t,out}))$ at free phase equilibrium. The gray line segments represent variation range of $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ in the entire positive or negative phase.

of a single rollout, we selected 10 consecutive rollouts. Similarly, in each rollout update, 1000 randomly selected samples of data are used for two nudge phases. Therefore, in total, 10000 samples are collected for the statistics of the advantage-r_{nudging} relationship. The effect of three different ϵ_{rev} are also investigated. The data for each ϵ_{rev} are aggregated into 16 advantage bins spanning $[-4, 4]$. Within each bin, for each ϵ_{rev}, the mean and standard deviation of extreme $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ are calculated. Extreme $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ means the ever reached most distant $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ value from zero. Intuitively, extreme $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ represent the tip of the gray line segments in Fig. 5. Fig. 6 shows the statistics of the extremes of $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ within each aggregation bin for 10000 samples in total.

From Fig. 6, we can see that the advantage-r_{nudging} relationship is approximately the same for different ϵ_{rev}. However, a critical difference is present in the region $\hat{A}_t \in [2.5, 4.0]$ in positive nudge phases: the $\epsilon_{\text{rev}} = 1.0$ setting gives significantly lower extreme $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ than $\epsilon_{\text{rev}} = 0.3$ and $\epsilon_{\text{rev}} = 0.7$.

To understand this phenomenon, consider the following equation in the positive nudge phase, and when the advantage is positive:

$$\frac{\partial \xi_{t,out,i}}{\partial t} = -\xi_{t,out,i} + \rho'(\xi_{t,out,i}) \sum_j w_{out,ij}\rho(\xi_{t,j}) + b_{out,i}\rho'(\xi_{t,out,i}) \tag{A-31}$$

$$- \beta \cdot \mathbf{1}_{(1-\epsilon_{\text{rev}}, 1+\epsilon)}(r_{t,\text{nudging}}(\xi_{t,out})) \cdot \frac{(a_{t,i} - \xi_{t,out,i})}{\sigma_i} \hat{A}_t$$

The nudge force is driving ξ_{out} away from the sample action. However, given $r_{t,\text{nudging}}(\xi_{t,out}) > 0$ and $\epsilon_{\text{rev}} = 1.0 \Leftrightarrow 1 - \epsilon_{\text{rev}} = 0$

$$\mathbf{1}_{[0,1+\epsilon)}(r_{t,\text{nudging}}(\xi_{t,out}))$$

cannot bound $r_{t,\text{nudging}}(\xi_{t,out})$ shift in the negative direction. $r_{t,\text{nudging}}(\xi_{t,out})$ can be indefinitely close to zero. For all other $\epsilon_{\text{rev}} < 1.0$, $r_{t,\text{nudging}}(\xi_{t,out})$ shift is bounded in the negative direction. The discussion about other scenarios: (positive nudge/negative advantage), (negative nudge/positive advantage), (negative nudge/negative advantage) is in Appendix C.

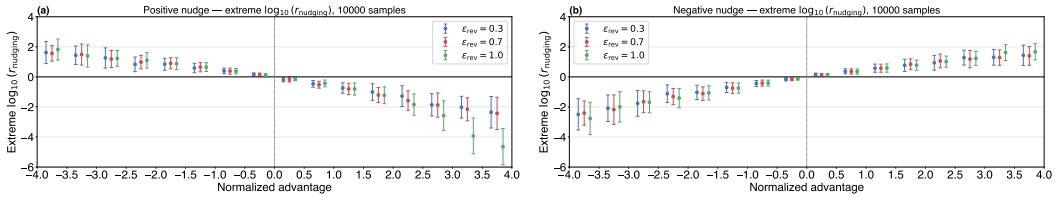

Figure 6: Statistics of the aggregated extreme $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ across 10 rollouts. It can be observed that in the left panel, in the normalized advantage range $[2.5, 4]$, the extreme $\log_{10}(r_{t,\text{nudging}}(\xi_{t,out}))$ is significantly lower for $\epsilon_{\text{rev}} = 1.0$ compared with two other cases.

E ABLATION STUDIES

E.1 EFFECT OF DIFFERENT REVERSE PPO RATIO CLIP COEFFICIENTS

Fig. 7 shows the rollout mean reward, KL-div, and the value MSE in the training process for three values of ϵ_{rev}. It can be observed that $\epsilon_{\text{rev}} = 0.3$ and $\epsilon_{\text{rev}} = 0.7$ achieve similar convergence speed, and they both converge to the optimum. $\epsilon_{\text{rev}} = 1.0$ failed to converge. Also, one version of $\epsilon_{\text{rev}} = 1.0$ results in a numerical stability issue in the training process, i.e., the EP network parameters all become NaN. Therefore, we only collect and plot the mean and standard deviation for 4 versions of $\epsilon_{\text{rev}} = 1.0$. From Fig. 7 (b), we can see spiking of per-update KL-div and frequent update rollback, which is indicated by zero value KL-div, which is a sign of excessively large update step. The cause of failed convergence is comprehensively discussed in the Methods section and Appendix C.

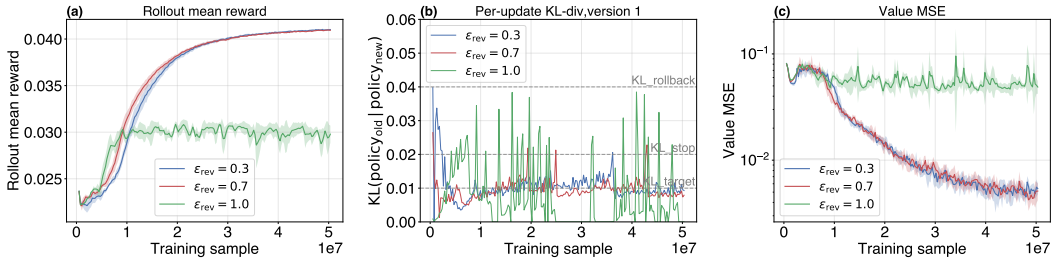

Figure 7: Convergence of stage-1 training for three values of ϵ_{rev}. The center line and shaded region indicate the mean and standard deviation across 5 versions for $\epsilon_{\text{rev}} = 0.3$ and $\epsilon_{\text{rev}} = 0.7$. For $\epsilon_{\text{rev}} = 1.0$, only 4 versions are aggregated because there is one version resulting in a numerical instability issue during training.

E.2 EFFECT OF $\frac{1}{\sigma}$ AND $\frac{1}{\sigma^2}$ IN THE OBJECTIVE GRADIENT

Fig. 8 shows the rollout mean reward, KL-div, and the value MSE in the training process when the output-gradient scaling uses $\frac{1}{\sigma}$ or $\frac{1}{\sigma^2}$. While $\frac{1}{\sigma}$ and $\frac{1}{\sigma^2}$ yield similar speed of mean reward improvement, and both of them approach the optimal policy, $\frac{1}{\sigma^2}$ leads to more frequent spikes in the KL-div graph, and the value MSE is not being minimized stably. This phenomenon implies an unstable policy network update. The cause is discussed in Appendix C. Therefore, we choose $\frac{1}{\sigma}$ for the proposed new objective gradient.

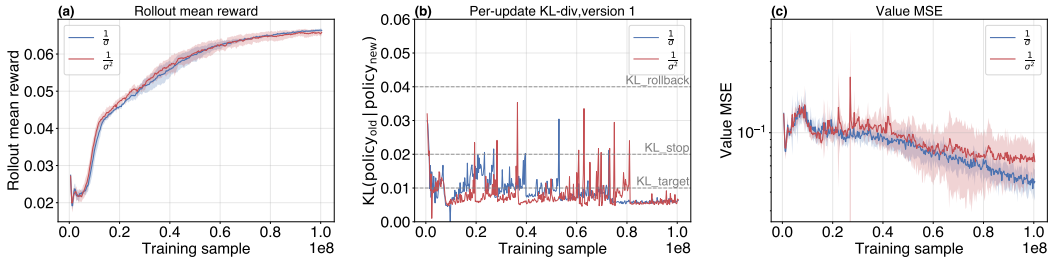

Figure 8: Convergence of stage-2 training for gradient scaling factor $\frac{1}{\sigma}$ and $\frac{1}{\sigma^2}$. The center line and shaded region indicate the mean and standard deviation across 5 versions.

E.3 EFFECT OF DYNAMIC AND STATIC GRADIENT MASK

Fig. 9 shows the rollout mean reward, KL-div, and the value MSE in the training process for the dynamic and static gradient mask. The definition of static gradient masks is as follows. When using

a static gradient mask, the objective gradient applied to the output neuron state $\xi_{t,out,i}$ is:

$$\mathbf{StaticMask}_t \frac{1}{|\mathcal{B}|} \frac{(a_{t,i} - \xi_{t,out,i})}{\sigma_i^2} \hat{A}_t \tag{A-32}$$

A constant gradient mask $\mathbf{StaticMask}_t = \mathbf{1}_{[0,1+\epsilon)}(r_t(\xi_{t,out}))$ or $\mathbf{StaticMask}_t = \mathbf{1}_{(1-\epsilon,\infty)}(r_t(\xi_{t,out}))$ is calculated based on the free phase equilibrium output neuron state before the nudge phase.

This constant mask is used in the nudge phase without changing. The static gradient masks mimic the backpropagation-style gradient masks, which are calculated only once. It can be clearly noticed that a static mask-based objective gradient cannot improve the policy. For a static gradient mask-based objective gradient, only 5×10^7 samples are trained.

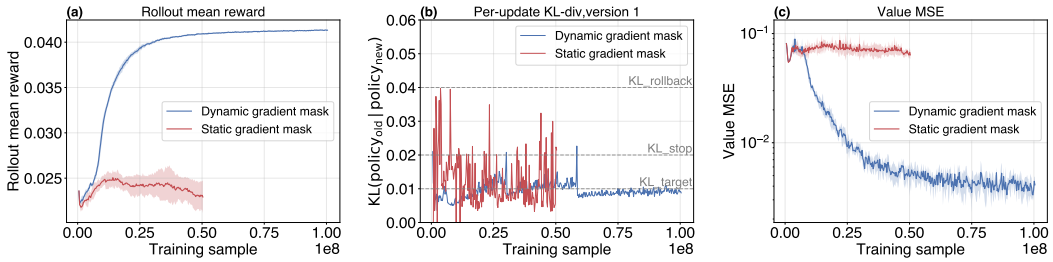

Figure 9: Convergence of stage-1 training for dynamic and static gradient mask. The center line and shaded region indicate the mean and standard deviation across 5 versions.

E.4 Effect of iDCT observation preprocessing

Fig. 10 shows the effect of converting the observation vector to a high-dimensional space through inverse discrete cosine transform. First, a higher input dimensionality increases the number of parameters in the first weight matrix (the weight matrix closest to the input), which yields better convergence speed, as shown in Fig. 10 (g). Second, we define the step to convergence as follows. Keep monitoring the $\mathrm{MaxStateChange} = \mathbf{max}(\Delta\xi_i)\ \forall i$ before and after each EP state update in relaxation, where i is the index of the neuron set of the entire network. If for the previous 5 consecutive steps, $\mathrm{MaxStateChange} < 10^{-4}$, the current step is marked as the step to convergence. In Fig. 10 (a),(b),(e),(f), it can be observed that, with iDCT preprocessing, the free phase of the policy network, the free and nudge phases of the value network requires a smaller number of steps to converge. Besides, Fig. 10 (c),(d) shows the percentage of samples that can converge within 50 relaxation steps. This percentage reduces faster when there is no iDCT preprocessing, which implies a long convergence process for a greater number of samples.

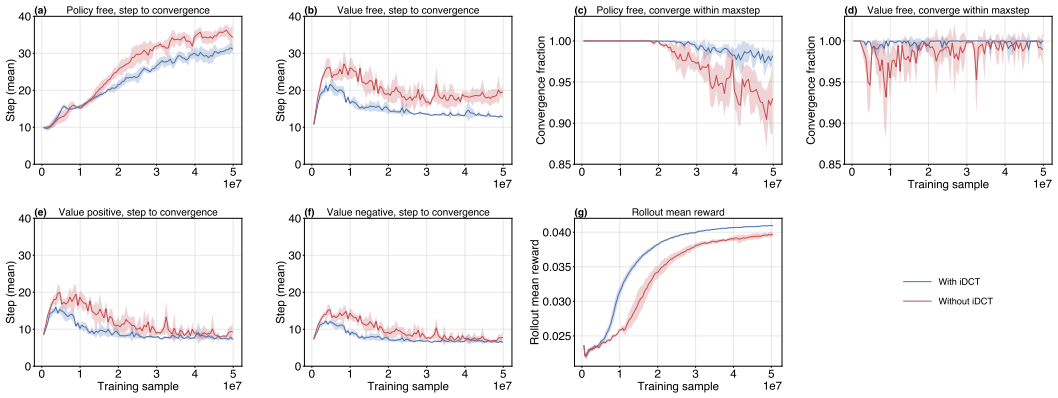

Figure 10: Step to convergence in the EP relaxation for both policy and value networks. The test is conducted in stage-1 training. The center line and shaded region indicate the mean and standard deviation across 5 versions. (c) and (d) show the percentage of the samples that converges within 50 steps. (g) indicates the convergence speed.

F EP-PPO ALGORITHM

Require: policy network $\mathbf{Policy}(s)$, value network $\mathbf{Value}(s)$, log-std vector $\log(\sigma)$
Require: number of environments N, rollout length T, discount factor γ, GAE parameter λ, policy learning rate η_{policy}, policy learning rate lower and upper bounds $\eta_{\text{policy,lower}}, \eta_{\text{policy,upper}}$, policy learning rate adaptation coefficient κ, value learning rate η_{value}, log-std learning rate η_{logstd}
Require: number of PPO epochs K_{epoch}, mini-batch size $|\mathcal{B}|$, target KL/early-stop KL/rollback KL $\text{KL}_{\text{target}}/\text{KL}_{\text{stop}}/\text{KL}_{\text{rb}}$
Require: EP nudge coefficient β_{ep}, policy free/nudge steps T^{π}_{free}, T^{π}_{+}, T^{π}_{-}, value free/nudge steps T^{V}_{free}, T^{V}_{+}, T^{V}_{-}
Require: iDCT matrix W_{iDCT}

Initialize policy network parameters, value network parameters, and log-std vector $\log(\sigma)$
Initialize SGD optimizer for policy network parameters with lr $= \eta_{\text{policy}}$, momentum $= 0.9$, weight decay $= 0$
Initialize SGD optimizer for value network parameters with lr $= \eta_{\text{value}}$, momentum $= 0.9$, weight decay $= 0$
Initialize Adam optimizer for $\log(\sigma)$ with lr $= \eta_{\text{logstd}}$, betas $= (0.9, 0.999)$, eps $= 10^{-8}$, weight decay $= 0$
Initialize raw observation running normalizer $\text{Normalizer}_{\text{raw}}$
Initialize iDCT observation running normalizer $\text{Normalizer}_{\text{iDCT}}$
Initialize N environments
Initialize $\text{SampleCounter} = 0$
while $\text{SampleCounter} < \text{MaxTrainingSamples}$ **do**

— Rollout collection —
  for $t = 0$ to $T - 1$ **do**
    for $n = 1$ to N **do**
      Observe $s_{n,t}$
      Sample $\varepsilon_{n,t} \sim \mathcal{N}(0, I)$
      Execute $a_{n,t} = \mathbf{Policy}(s_{n,t}) + \sigma \odot \varepsilon_{n,t}$
      Step environment and store data tuple into buffer

$$(s_{n,t},\, a_{n,t},\, r_{n,t},\, \log(\pi_{\text{rollout}}(a_{n,t}|s_{n,t})),\, d_{n,t},\, \text{fall}_{n,t},\, \mathbf{Value}(s_{n,t}),\, \mathbf{Value}(s_{n,t+1}))$$

▷ See Appendix C

    end for
  end for
  $\text{SampleCounter} \mathrel{+}= N \cdot T$

— Advantage and return computation —
  Compute estimated $\left\{(\mathbf{Return}(s_{n,t}), \mathbf{Advantage}(s_{n,t}, a_{n,t}))\right\}_{n=1,t=1}^{N,T}$, noted as $\left\{(\hat{R}_{n,t}, \hat{A}_{n,t})\right\}_{n=1,t=1}^{N,T}$ using GAE and data in the buffer. Here the input to GAE is $\left\{(r_{n,t},\, d_{n,t},\, \text{fall}_{n,t},\, \mathbf{Value}(s_{n,t}),\, \mathbf{Value}(s_{n,t+1})\right\}_{n=1,t=1}^{N,T}$.
  Flatten all $N \times T$ samples into

$$\mathcal{D} = \left\{(s_t,\, a_t,\, \log(\pi_{\text{rollout}}(a_t|s_t)),\, \hat{A}_t,\, \hat{R}_t)\right\}_{t=1}^{NT}$$

  Apply normalization and iDCT to observations:

$$s_t \leftarrow \text{Normalizer}_{\text{raw}}(s_t)$$

$$s_t \leftarrow \text{Normalizer}_{\text{iDCT}}\big(W_{\text{iDCT}} \cdot s_t\big)$$

  Update running normalizers $\text{Normalizer}_{\text{raw}}$, $\text{Normalizer}_{\text{iDCT}}$
  Normalize advantages:

$$\hat{A}_t \leftarrow \frac{\hat{A}_t - \mu_A}{\sigma_A + 10^{-8}}, \quad \forall t$$

where μ_A, σ_A are the mean and standard deviation of the raw advantages over the rollout

Snapshot policy network parameter and log-std vector($\mathbf{Policy}_{\mathrm{snap}}, \log(\sigma)_{\mathrm{snap}}$) $\leftarrow$ ($\mathbf{Policy}, \log(\sigma)$), where $\mathbf{Policy}_{\mathrm{snap}}$
caches old means $\mu_t^{\mathrm{old}} \leftarrow \mathbf{Policy}(s_t)$

— Policy update with EP three-phase —
for $k = 1$ to K_{epoch} **do**
Shuffle $\mathcal{D}$
for each mini-batch $\mathcal{B} \subset \mathcal{D}$ **do**
Free phase:
Initialize policy neurons ξ;
Relaxation for T_{free}^{π} steps with $\beta = 0$;
Obtain fixed point ξ^*
Positive nudge:
Initialize policy neurons $\xi \leftarrow \xi^*$
Relaxation for T_{+}^{π} steps with loss gradient in Equation 11 and $\beta = \beta_{\mathrm{ep}}$
Obtain fixed point $\xi^{+\beta_{\mathrm{ep}}}$
Negative nudge:
Initialize policy neurons $\xi \leftarrow \xi^*$
Relaxation for T_{-}^{π} steps with loss gradient in Equation 11 and $\beta = -\beta_{\mathrm{ep}}$
Obtain fixed point $\xi^{-\beta_{\mathrm{ep}}}$
Compute policy network parameter gradient by using $\xi^{+\beta_{\mathrm{ep}}}, \xi^{-\beta_{\mathrm{ep}}}, \beta_{\mathrm{ep}}$ with Equation 7 and SGD step
Adam step on $\log(\sigma)$ with gradient in Equation A-27 ▷ log-std vector update
end for
Estimate analytic mean KL between current and snapshot policy:

$$\widehat{\mathrm{KL}} = \frac{1}{2|\mathcal{D}|} \sum_{t} \sum_{i}^{D_{\mathrm{action}}} \frac{\|\mathbf{Policy}(s_t)_i - \mu_{t,i}^{\mathrm{old}}\|^2}{\sigma_i^2}$$

The change of σ is ignored in the KL-div calculation because σ only changes slightly for each rollout.
if $\widehat{\mathrm{KL}} > \mathrm{KL}_{\mathrm{stop}}$ **then**
break ▷ KL early stopping of policy epochs
end if
end for

— Value update with EP three-phase —
for $k = 1$ to K_{epoch} **do**
Shuffle $\mathcal{D}$
for each mini-batch $\mathcal{B} \subset \mathcal{D}$ **do**
Free phase:
Initialize value neurons ξ_v;
Relaxation for T_{free}^{V} steps with $\beta = 0$;
Obtain fixed point ξ_v^*
Positive nudge:
Initialize value neurons $\xi_v \leftarrow \xi_v^*$
Relaxation for T_{+}^{V} steps with loss in Equation A-12 and $\beta = \beta_{\mathrm{ep}}$
Obtain fixed point $\xi_v^{+\beta_{\mathrm{ep}}}$
Negative nudge:
Initialize value neurons $\xi_v \leftarrow \xi_v^*$
Relaxation for T_{-}^{V} steps with loss in Equation A-12 and $\beta = -\beta_{\mathrm{ep}}$
Obtain fixed point $\xi_v^{-\beta_{\mathrm{ep}}}$
Compute value network parameter gradient by using $\xi_v^{+\beta_{\mathrm{ep}}}, \xi_v^{-\beta_{\mathrm{ep}}}, \beta_{\mathrm{ep}}$ with Equation 7 and SGD step
end for
end for

— KL safeguard and adaptive LR —

if $\widehat{\mathrm{KL}} > \mathrm{KL}_{\mathrm{rb}}$ **then**
  Restore $(\mathbf{Policy},\ \log(\sigma)) \leftarrow (\mathbf{Policy}_{\mathrm{snap}},\ \log(\sigma)_{\mathrm{snap}})$ ▷ policy and log-std rollback
  $\eta_{\mathrm{policy}} \leftarrow \eta_{\mathrm{policy}} / \kappa^2$
else if $\widehat{\mathrm{KL}} > 2\,\mathrm{KL}_{\mathrm{target}}$ **then**
  $\eta_{\mathrm{policy}} \leftarrow \eta_{\mathrm{policy}} / \kappa$
else if $\widehat{\mathrm{KL}} < 0.5\,\mathrm{KL}_{\mathrm{target}}$ **then**
  $\eta_{\mathrm{policy}} \leftarrow \kappa\,\eta_{\mathrm{policy}}$
end if
$\eta_{\mathrm{policy}} \leftarrow \mathbf{Clamp}(\eta_{\mathrm{policy}}, \eta_{\mathrm{policy,lower}}, \eta_{\mathrm{policy,upper}})$ ▷ Clip policy learning rate
end while

G LIST OF HYPER-PARAMETERS

Table 6: List of hyper-parameters. If a hyper-parameter is different in stage-1 and stage-2, the value in each stage will be displayed individually in the form (**1:** value1, **2:** value2).

Hyper-parameter	**Value**	**Description**
N	(**1:** 1024, **2:** 2048)	Number of simulation environments
-	2000	Maximum episode length
$v_{x,\text{upper}}$	$0.5\,\text{m/s}$	Upper bound of local x-directional target velocity
$\mathbf{ResLimit}$	1.0 rad	Residual angle integration bound
$\mathbf{TorqueLimit}$	$33.5\,\text{N m}$	Robot actuator torque bound
$\mathbf{SideLength_{low}}$	$0.3\,\text{m}$	Terrain box side length lower bound
$\mathbf{SideLength_{high}}$	$0.5\,\text{m}$	Terrain box side length upper bound
MaxTrainingSamples	10^8	Total training samples
γ	0.99	RL discount factor
λ	0.95	GAE lambda
T	(**1:** 256, **2:** 128)	Per-env rollout length
K_{epoch}	10	Number of training epochs for policy and value networks
-	4	Number of training mini-batches in each epoch for policy and value networks
$\text{KL}_{\text{target}}$	0.01	Target KL-divergence
KL_{stop}	0.02	Early-stop KL-divergence
KL_{rb}	0.04	Rollback KL-divergence
-	17.03	Entropy scheduler initial entropy
-	17.03	Entropy scheduler final entropy
k_{entropy}	0.01	Entropy loss coefficient
η_{logstd}	3×10^{-4}	Constant log-std learning rate
$\eta_{\text{policy,initial}}$	0.1	Initial policy learning rate
$\eta_{\text{policy,lower}}$	10^{-6}	Policy learning rate lower bound
$\eta_{\text{policy,upper}}$	10	Policy learning rate upper bound
κ	1.5	Policy learning rate adaptation coefficient
η_{value}	0.1	Constant value learning rate
-	Float32	EP network datatype for neuron state and model parameters
ϵ	0.2	PPO ratio clip coefficient
ϵ_{rev}	0.7	Reverse PPO ratio clip coefficient
α_w	0.5	EP weight initialization scale
T^{π}_{free}, T^{π}_{+}, T^{π}_{-}	$30, 20, 10$	Policy free/nudge steps
T^{V}_{free}, T^{V}_{+}, T^{V}_{-}	$25, 15, 10$	Value free/nudge steps
β_{ep}	0.1	EP beta
ϵ_{ep}	1.0	EP state update coefficient