Non-conflicting Energy Minimization in Reinforcement Learning based Robot Control

Skand Peri **Akhil Perincherry**[*] **Bikram Pandit**[*] **Stefan Lee**
Oregon State University
Project Page: https://pvskand.github.io/projects/PEGrad

Abstract: Efficient robot control often requires balancing task performance with energy expenditure. A common approach in reinforcement learning (RL) is to penalize energy use directly as part of the reward function. This requires carefully tuning weight terms to avoid undesirable trade-offs where energy minimization harms task success. In this work, we propose a hyperparameter-free gradient optimization method to minimize energy expenditure without conflicting with task performance. Inspired by recent works in multitask learning, our method applies policy gradient projection between task and energy objectives to derive policy updates that minimize energy expenditure in ways that do not impact task performance. We evaluate this technique on standard locomotion benchmarks of DM-Control and HumanoidBench and demonstrate a reduction of 64% energy usage while maintaining comparable task performance. Further, we conduct experiments on a Unitree GO2 quadruped showcasing Sim2Real transfer of energy efficient policies. Our method is easy to implement in standard RL pipelines with minimal code changes, is applicable to any policy gradient method, and offers a principled alternative to reward shaping for energy efficient control policies.

Keywords: Energy-efficient Locomotion, Reinforcement Learning,

1 Introduction

Untethered robots are inherently constrained by their battery capacity limiting their deployment duration. For example, the Unitree Go2 [1] typically operates for only 1–4 hours per charge under low-speed locomotion using its factory controllers, with charging times ranging from 1–2 hours. To maximize operational time, control policies must minimize energy expenditure while still ensuring task completion – any excess energy use shortens battery life while insufficient effort may result in higher rates of task failure. Furthermore, high-energy behaviors can pose safety risks to the robot and its environment, such as excessive actuator wear or destabilizing interactions with the ground.

One of the most common and increasingly popular ways of learning robot control policies has been using reinforcement learning (RL) [2, 3, 4, 5, 6, 7, 8] – especially for complex and highly dynamic embodiments like legged robots. During policy training, energy minimization is typically incorporated into the reward function through penalties on joint torques [6, 8] or mechanical work [9, 10]. A common formulation is a weighted sum: $r = r_{\text{task}} + \lambda r_{\text{energy}}$, where r_{task} is the task specific reward and r_{energy} is the energy penalty. The λ coefficient then becomes a critical hyperparameter for balancing between these (often competing) objectives. However, different embodiments and tasks naturally demand different effort, e.g., running being more energetic than standing. Consequentially, the performance of learned policies can be highly sensitive to the choice of λ, making it difficult to tune in practice. As shown in the left plot in Figure 1, different values of λ offer different trade-offs between a trained policy's average return and overall motor torque for a quadruped running task. While λ's of 0, 0.001, and 0.01 all produce similar returns (though differing torques), higher values begin to significantly degrade task performance as energy minimization starts to dominate.

[*]Equal contribution.

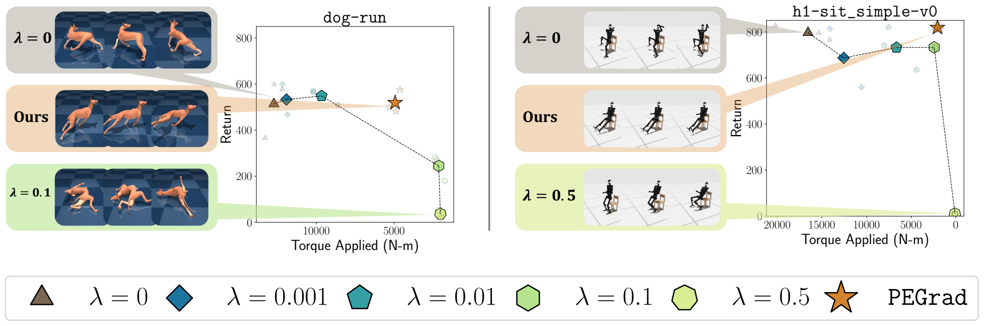


Figure 1: RL control policies often optimize a weighted combination of task reward and energy penalties, i.e., $r_{\text{task}} + \lambda r_{\text{energy}}$. However, tuning the weighting factor λ is challenging due to high variability in its optimal value across tasks, environments, and embodiments. *(Left)* When a Soft Actor Critic (SAC) agent is trained on the `dog-run` task, λ=0.01 and λ=0.1 result in significantly different performance – with the policy at λ=0.1 achieving low returns by crawling rather than running. However, λ=0.1 works well in the less dynamic `dog-walk` environment *(Not shown)*. *(Right)* For a humanoid `sitting` task, both λ=0.01 and 0.1 yield policies that are equally energy-efficient and task-effective, showcasing the inter-environment variability. In both cases, our proposed hyperparameter-free method, PEGrad (★), leads to performant and energy efficient policies. (The lightly shaded markers represent the checkpoint with the highest average evaluation score over 50 episodes, while the solid markers indicate the overall average value across all runs.)

Rather than tuning a trade-off between energy expenditure and task performance, we would ideally like to specify an ordering of these objectives. That is to say, we would like robots to expend the least amount of energy ***necessary*** to successfully complete the task – adapting energy use to task demands without requiring per-task manual tuning. To address this, we propose PEGrad – a hyperparameter-free method that trains policies that simultaneously achieve high task performance and low energy expenditure. As shown by the orange stars (★) Figure 1, PEGrad-based policies can automatically find performant policies with low energy expenditure across different tasks and embodiments.

Our key idea is to formulate policy training as a multi-objective optimization problem and derive a descent direction for policy parameters which minimize energy by moving along approximate level-sets of the reward function. Specifically, we modify the gradients of energy with respect to the policy parameters to be orthogonal to the task reward gradient at each time step of optimization. We conduct experiments on various locomotion benchmarks such as `DM-Control` suite, `HumanoidBench`, and `IssacLab` that show the efficacy of PEGrad. Specifically, we show an average of 64% energy-efficiency on these benchmarks compared to vanilla RL policy ($\lambda = 0$). Finally, we transfer a learned policy to a real robot to demonstrate real-world improvements to battery life from running our policies.

Contributions. We summarize our main contributions:

- We propose PEGrad, a hyperparameter-free gradient optimization method that minimizes energy expenditure without conflicting with task performance in policy gradient RL methods.

- We experiment in two simulated benchmarks, DM-Control [11] and HumanoidBench [12], and show that PEGrad decreases energy expenditure by 64% while maintaining task performance.

- We demonstrate the Sim2Real transfer of PEGrad on Unitree GO2 robot and evaluate energy efficiency in terms of the current drawn from the battery and the net torque applied by the robot while executing Standing and Walking (SaW) policies.

2 Related Works

Multi-objective optimization: Multi-objective optimization (MOO) studies the problem of optimizing a set of potentially conflicting objectives. We focus on MOO methods that explicitly manipulate gradients to mitigate negative task interference [13, 14, 15, 16, 17]. GradNorm [18] balances magnitudes of task gradients by modulating them based on task training rates. However, it disregards gradient directional conflicts and relies on hyperparameter tuning. PCGrad [16] modifies task gradients by conditionally projecting a task's gradient onto the orthogonal sub-space of another task's gradient only when a conflict is detected based on negative cosine similarity, and does not consider orderings for the objectives. In our work, we project energy minimization gradients onto the orthogonal sub-space of reward maximization gradients at all times thereby prioritizing reward maximization over energy minimization. We show empirically that directly applying PCGrad-style conditional projection leads to sub-optimal policies in Sec. 4.1. Grad-Similarity [19] uses cosine similarity between task gradient vectors to estimate task-relatedness. However, the objective of Grad-Similarity is to select auxiliary tasks that can optimize a single main task objective whereas our work attempts to jointly optimize multiple task objectives. CAGrad [17] uses a hyperparameter to find a vector that maximizes worst-case improvement while staying within a ball around the average gradient of tasks. Our method is hyperparameter free and does not explicitly constrain the resulting vector to be proximate to the average gradient. GradDrop [20] randomly drops conflicting gradient components based on their signs to reduce interference. However, in contrast to our work, GradDrop does not consider gradient magnitudes and task importance. Xu et al. [21] perform adaptive gradient updates to ensure that no single objective dominates, whereas, our work assumes a priority ordering where gradients of the task reward are more important than that of energy.

Energy optimization in Legged Robots: Nai et al. [22] train a surrogate model to predict energy consumption by deploying a pre-trained policy and collecting real-world power usage data. This surrogate model is then used to define an additional energy-based reward, which guides the fine-tuning of the original policy. The authors adopt an iterative refinement loop involving real-world data collection, surrogate model training, and policy fine-tuning. In contrast, our approach focuses on directly optimizing for energy efficiency in simulation, followed by transfer to the real world—eliminating the need for a learned energy model as well as an iterative refinement process. Fu et al. [23] train reinforcement learning policies to minimize energy—formulated as mechanical work—alongside task and survival rewards, and observe that the resulting behaviors resemble natural gaits in biological animals. While they manually tune the reward weights, we propose to automatically learn a policy that minimizes energy while preserving task performance. Mahankali et al. [24] formulate the energy-performance trade-off as a constrained optimization problem by maintaining two separate policies: one for the combined task and energy objective (π) and another for the task objective alone (π'). While π' is optimized conventionally, the updates to π are penalized whenever its task reward falls below that of π', with the penalty scaled by a learnable parameter α. This parameter α is dynamically adjusted to keep the returns of π and π' close. Although their problem setting aligns with ours, we take a different approach: instead of maintaining two policies, we introduce two critics and optimize a single policy.

3 Methodology

We formulate our problem as a Multi-Objective Markov Decision Process (MOMDP) [25] defined by the tuple $(S, A, T, R, \gamma, \mu_0)$ where S and A are the state and action spaces, $T(s'|s,a)$ is the transition dynamics, $R(s)$ is a vector-valued reward function representing scalar rewards for each objective, γ is the discount factor, and $\mu_0(s)$ is the start state distribution. In our setting, we consider the reward function to be 2-dimensional – returning both a task reward $r(s)$ and energy consumption $e(s)$ at state s. Our overall goal will be to minimize energy consumption without sacrificing task reward.

3.1 Preliminaries

While our proposed method is applicable to any policy-gradient algorithm, we present results on two popular actor-critic [26] algorithms – Soft Actor Critic (SAC) [27] and Proximal Policy Optimization (PPO) [28] – and focus our discussion here on SAC to describe our approach. For a matching discussion of PPO, see the Appendix 8.1.

Soft Actor Critic (SAC). In standard single-objective settings, SAC aims to maximize both expected returns and policy entropy. Following the actor-critic framework, SAC learns a scalar action-value function $Q_\phi(s,a)$ and a stochastic policy $\pi_\theta(a|s)$. The critic network $Q_\phi(s,a)$ is trained to minimize

$$\mathscr{L}_Q = \left(Q_\phi(s,a) - \left(r + \gamma Q_{\bar{\phi}}(s',a') - \alpha \log \pi_\theta(a' \mid s')\right)\right)^2 \tag{1}$$

where $a' \sim \pi_\theta(.|s')$, α is an entropy coefficient, and $Q_{\bar{\phi}}$ is the target Q-network implemented as an exponential moving average of Q_ϕ. The policy function $\pi_\theta(a|s)$ is trained to minimize

$$\mathscr{L}_\pi = \mathbb{E}_{a\sim\pi_\theta}\left[\alpha \log \pi_\theta(a \mid s) - Q_\phi(s,a)\right]. \tag{2}$$

Both the critic and policy networks are trained simultaneously.

Multi-Objective SAC. For our dual objective setting, we employ two critic networks – one for estimating the task action-value function $Q^r_\phi(s,a)$ and the other for the energy action-value function $Q^e_{\phi_e}(s,a)$. Each critic can be trained independently following Eq. 1 for rewards and energy consumption. These critics can share parameters; however, we implement them independently here.

For deriving a policy in multi-objective settings, a common approach is to introduce (or learn) a utility function U that maps the vector-valued Q-function to a scalar by encoding some notion of the relative importance of each objective. This reduces the policy-learning problem back to that of a single-objective MDP where the resulting utility can be maximized. A common choice of utility function when it comes to energy minimization is a simple linear combination parameterized by a trade-off parameter λ. For SAC, this corresponds to the following actor objective:

$$\mathscr{L}_\pi = \mathbb{E}_{a\sim\pi_\theta}\left[\alpha \log \pi_\theta(a \mid s) - Q^r_\phi(s,a) + \lambda Q^e_{\phi_e}(s,a)\right] \tag{3}$$

Larger values of λ will more strongly move the policy's distribution away from high energy states and actions. However, it is often difficult to set the trade-off parameter λ *a priori* for a new task. Too high and the policy may fail to learn or produce sluggish behaviors. Too low and energy may not be effectively optimized. Furthermore, optimizing combinations of conflicting objectives can result in sub-optimal learning [16] – which certainly include task reward and energy expenditure for highly dynamic robot control tasks like locomotion.

3.2 Projecting Energy Gradients (PEGrad)

To alleviate this issue, we introduce our method **P**rojecting **E**nergy **Grad**ients (PEGrad). To motivate our proposed method, we consider a multi-objective setting in which energy minimization is a subordinate objective that should be optimized *only* when doing so does not interfere with reward maximization. To start, let us separate the actor objective into two losses corresponding to the task reward and energy reduction, denoting these as

$$\mathscr{L}_R(\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\alpha \log \pi_\theta(a \mid s) - Q^r_\phi(s,a)\right] \quad \text{and} \quad \mathscr{L}_E(\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\alpha \log \pi_\theta(a \mid s) + Q^e_{\phi_e}(s,a)\right] \tag{4}$$

For a small change d in policy parameters θ, the reward objective for the resulting policy under a first-order Taylor approximation can be written as

$$\mathscr{L}_R(\theta + d) \approx \mathscr{L}_R(\theta) + g_R^T d \tag{5}$$

where g_R is the gradient of the reward loss with respect to network parameters evaluated at θ, i.e., $g_R = \nabla_\theta \mathscr{L}_R(\theta)$. From this, standard gradient descent algorithms setting $d = -\alpha g_R$ can be derived where α is a learning rate hyperparameter. More to our point however, this approximation also implies that a small change in a direction orthogonal to g_R results in *no* change to the reward loss function – corresponding to movement along a level-set of the approximated reward loss hyperplane. Naturally, this is only valid in a small region around θ where the approximation holds.

This suggests a straight-forward algorithm in which the energy consumption loss is minimized only by shifting parameters in this orthogonal space of the reward loss gradient. By the same reasoning as above, changing θ by adding $-g_E = -\nabla_\theta \mathscr{L}_E(\theta)$ would move to minimize energy; however, $-g_E$ may have components that would also modify $\mathscr{L}_R$. Instead, we consider the orthogonal projection of g_E onto g_R denoted as $g_{E_{\perp R}}$, taking the overall descent direction d as

$$d = -\alpha g_R - \beta\, g_{E_{\perp R}} = -\alpha g_R - \beta\left(g_E - \frac{g_R^T g_E}{g_R^T g_R} g_R\right) \tag{6}$$

Rather than tuning β as an independent learning rate hyperparameter, we define it adaptively as

$$\beta = \alpha * \min\left(1, \frac{\| g_R \|_2}{\| g_{E_{\perp R}} \|_2}\right) \tag{7}$$

such that the norm of $\beta g_{E_{\perp R}}$ is no greater than the norm of αg_R. We provide an empirical justification for this choice, finding it to work well across multiple settings.

Implementing `PEGrad`. Alg. 1 outlines a practical implementation of our proposed approach. After performing backward passes for each policy loss component, energy loss gradients are directly adjusted via projection and conditionally rescaled. Rather than setting α as a learning rate directly, we pass the updated gradient direction $g_R + g_{E_{\perp R}}$ to any choice of optimizer. This is applied at each step of training. Applying this algorithm represents a relatively small code change in existing RL frameworks but does incur the cost of a second backward pass to compute g_R and g_E separately.

Algorithm 1 `PEGrad`

```
Require: Policy ($\pi$) parameters $\theta$, task minibatch $\mathscr{B} = \{\mathscr{T}_k\}$
1: $g_R \leftarrow \nabla_\theta \mathscr{L}_R(\theta)$                                   ▷ Compute task reward loss gradient from batch
2: $g_E \leftarrow \nabla_\theta \mathscr{L}_E(\theta)$                                   ▷ Compute energy expenditure loss gradient from batch
3: $g_{E_{\perp R}} \leftarrow g_E - \frac{g_R^T g_E}{g_R^T g_R} g_R$                     ▷ Compute orthogonal projection
4: if $\|g_{E_{\perp R}}\|_2 > \|g_R\|_2$ then
5:     $g_{E_{\perp R}} = g_{E_{\perp R}} \frac{\|g_R\|_2}{\|g_{E_{\perp R}}\|_2}$        ▷ Rescale if larger norm than reward gradient
6: end if
7: return update $\Delta\theta = g_R + g_{E_{\perp R}}$                                   ▷ Pass returned value as gradient to optimizer
```

4 Experiments

We center our experiments and discussion around the following questions: **(Q1)** Can `PEGrad` minimize energy while retaining task performance across various environments and tasks? (Sec. 4.1); and **(Q2)** Can we use `PEGrad` to reduce battery usage on a real robot? (Sec. 4.2)

Formulation of energy. Prior works have used various formulations of energy such as torque penalty and mechanical work. In our experiments, we choose the sum of absolute torques applied to actuated motors as a proxy for energy. Concretely, we compute the energy function as $e(s) = \sum_{m=1}^{M} |\tau_m|$ where M is the total number of actuated motors of the robot and τ_m is the torque applied to the mth motor. This formulation aims to minimize battery current draw rather than system energy. In simulation, current I is unmeasurable but proportional to motor torque τ, and taking $|\tau|$ captures draw regardless of direction. On contrary, mechanical power $(\tau.\omega)$ depends on the velocity, and can underestimate I during high-torque, low-motion phases. Thus, $\sum|\tau|$ better reflects battery load. However, we note that this ignores differences in motors and gear ratios that impact current draw. If these are heterogeneous and known for an embodiment, appropriate coefficients may be included.

4.1 Simulation Experiments

Setting. For our simulation experiments, we use two environments based on the MuJoCo [29] physics simulator – DM-Control suite [11] and HumanoidBench [12].

`DM-Control`: We consider two robot embodiments `Quadruped` and `Dog` and conduct our experiments on `Quadruped-{Walk, Run}`, and `Dog-{Stand, Walk, Run, Trot}`. Tasks on the `Dog` embodiment are the most challenging in the suite owing to their 228-dim state space and 38 dimensional action space. Policy actions correspond directly to motor torques.

`HumanoidBench`: The `HumanoidBench` benchmark consists of 27 whole body control tasks. We focus on four fundamental locomotion tasks `{Stand, Walk, Run, Sit}`, all of them trained on Unitree H1 humanoid robot in simulation. All the tasks have state dimension of 51 and action dimension 19. Policy actions correspond directly to motor torques.

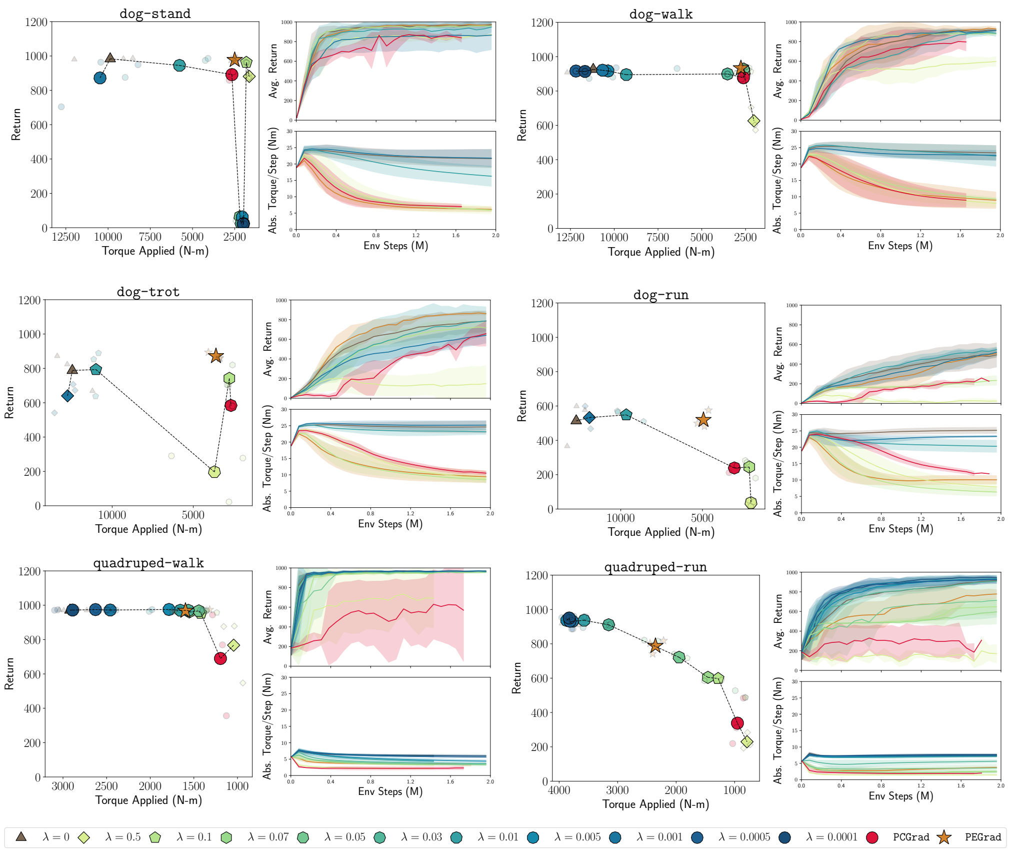


Figure 2: `DMControl Suite` Results: We show results on six tasks from `DMControl` suite [11]. Low applied torque and higher returns are better.Across all tasks, `PEGrad` achieves high task performant policies that are also energy-efficient. For 4 out of 6 tasks, `PEGrad` achieves results beyond the Pareto front identified by adjusting λ.

For the experiments on `DM-Control` suite and `HumanoidBench` we use SAC as the base RL algorithm that is implemented using LeanRL [30] (a PyTorch library based on CleanRL [31]). A list of hyperparameters and network architectures for critics and actor is described in Appendix 8.4.

Baselines. To validate our method, we compare `PEGrad` against the following baselines:

- `Base`: The base SAC implementation has a single critic and is trained on default environment reward functions. For `DM-Control` this corresponds to an energy unconstrained policy (denoted as λ=0), whereas `HumanoidBench` includes some default energy penalties (denoted as `Base`).
- `Multi-Objective (`λ`=`X`)`: This baseline includes two critics as described in Sec. 3.1 with a factor λ in the policy loss to trade off between task reward and energy efficiency as in Eq. 3. We use energy as defined above for all environments and remove default energy penalties from `HumanoidBench` rewards. We select $\lambda \in \{0.001, 0.01, 0.1, 0.5\}$ for our experiments.
- `PCGrad`$^+$ [16]: While originally proposed for multi-objective problems where all objectives are equally important, we adapt PCGrad to our setting where reward takes precedence over energy. The resulting method will first subtract the parallel component of g_E when g_E and g_R are conflicting i.e $g_R^\top g_E < 0$, but otherwise aggregates both gradients directly. We build on the public PyTorch implementation of this work [32].

Results. For each task setting, we provide three result plots – (i) a *Pareto front* showing our energy formulation vs. return for converged policies; (ii) a *sample efficiency* plot showing environment steps vs. return; and (iii) an *energy efficiency* plot showing environment steps vs. our energy formulation.

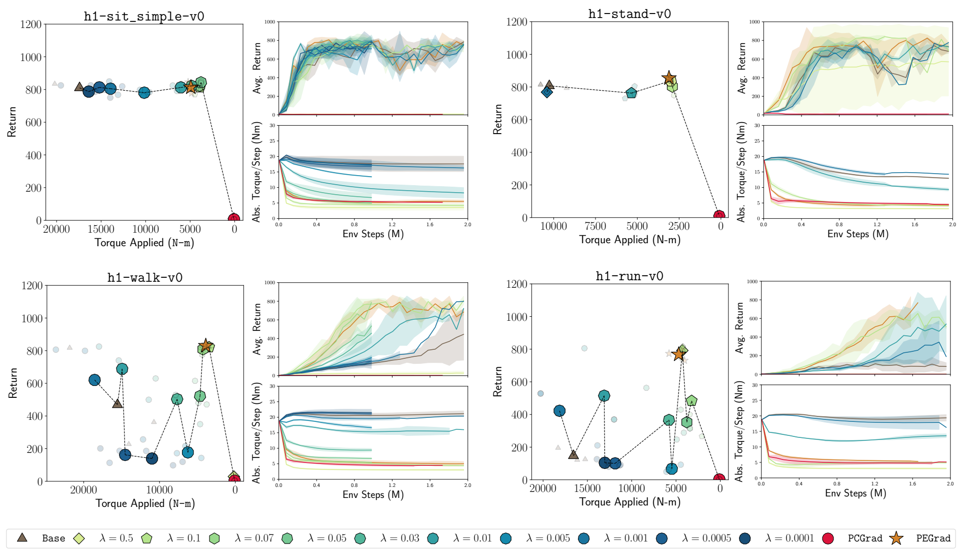


Figure 3: HumanoidBench Results: We show results on four tasks from HumanoidBench suite [12]. Low applied torque and higher returns are better. Across all tasks, PEGrad achieves highly performant policies that are also energy-efficient with PEGrad. Further, energy minimization also improves sample-efficiency on h1-run-v0 and h1-walk-v0 tasks.

All results shown are run for 1.5-2M steps and run for 3 seeds. Shaded areas in (ii)/(iii) are 95% bootstrapped CIs. For (i), mean results are plotted with individual seeds as shaded markers.

We report results for the DMControl suite of tasks in Fig. 2. We observe that PEGrad consistently produces energy efficient policies – achieving high rewards on-par with the unconstrained Base ($\lambda = 0$) policy while executing with significantly reduced torques. Interestingly, we observe that on both the quadruped and the more energetic dog-run and dog-trot tasks, PCGrad$^+$ leads to sub-optimal policies – sacrificing substantial returns for gains in energy efficiency.

We report results for the HumanoidBench suite of tasks in Fig. 3. We observe PEGrad not only leads to lower-energy and highly-performant policies but also shows *significant* sample-efficiency gains compared to the Base on run and walk tasks. The $\lambda = 0.1$ models also exhibit this phenomenon – indicating that minimizing energy objectives can lead to improved sample-efficiency in some RL tasks. Interestingly, we find that PCGrad$^+$ over-optimizes energy to the point of achieving nearly no return for any of the four tasks. The corresponding policy effectively falls without much actuation. Speculatively, we attribute this to PCGrad$^+$'s lack of any adaptive gradient scaling analogous to β.

4.2 Sim2Real on Go2 Quadruped

Experimental Setting. For our Sim2Real experiments, we train a standing and walking (SaW) controller for the Unitree Go2 quadruped in simulation using IsaacLab [33] and deploy it in the real world. We train using PPO [34] on flat terrain with standard domain randomization, and adopt the Adversarial Motion Prior (AMP) framework [35] for ”natural-looking” gaits which has been adopted extensively in robotic settings [36, 37, 38, 39, 40, 41]. AMP models a motion prior over a set of demonstration trajectories by training a discriminator that learns to distinguish between real trajectories and the robot's generated motions. The robot's control policy is trained in an adversarial fashion to this discriminator, thereby encouraging the robot to produce motions that are similar to the collected trajectories. To build a dataset of demonstrations, we use the factory-issued Go2 controller to collect a 90 second sequence of standing and walking trajectories. The trajectories consist of 24 dimensional states (joint positions and velocities) and 12-dimensional actions (joint-delta positions with respect to the canonical pose of the robot) recorded at 50Hz. During training, the robot's policy receives two rewards: one for tracking desired velocities (task performance), and another *style reward* from the AMP discriminator, encouraging gaits similar to the factory-issued controller. We provide more details about the specific architecture and details of AMP in Section 8.3.

SaW Controller	Standing		Walking	
	Current Drawn (mA)	Net Torque Applied (Nm)	Current Drawn (mA)	Net Torque Applied (Nm)
Factory	4.029 ±0.005	3.97 ±0.001	6.46 ±0.19	4.58 ±0.04
AMP+PPO $\lambda = 0$	3.473 ±0.005	3.47 ±0.083	✗	✗
AMP+PPO $\lambda = 0.0002$	**2.389 ±0.170**	**2.52 ±0.006**	7.04 ±0.90	4.68 ±0.01
AMP+PPO PEGrad	**2.533 ±0.022**	**2.45 ±0.105**	**5.65 ±0.45**	**3.94 ±0.01**

Table 1: Current and Torque usage in the real-world: We compare Unitree's Factory controller and AMP+PPO λ baselines against PEGrad for Standing and Walking tasks and report current drawn and net torque applied. We find that PEGrad is ~20% more efficient than the tuned multi-objective AMP+PPO λ=0.0002 on the task of walking and has a comparable performance on standing.

For the standing task, we command 0 velocity and compute metrics over 20 seconds. For the walking task, we command the robot with a velocity of 0.5m/s in the forward direction to cover a distance of 12 feet ($\sim$ 3.66m). For both, we report the current drawn as well as the net absolute torque applied in that time. For standing, we allow policies an initial burnin period during which we do not collect metrics. For walking, we likewise ignore the initial and final period where the commanded speed of the robot changes due to start-up and slow-down. These filtering steps are to account for input delays and the effects of switching from default to learned controllers.

Baselines. We compare the energy efficiency of PEGrad against the factory-issued Go2 controller (denoted as Factory) as well as AMP+PPO λ baselines with $\lambda = 0$ and $\lambda = 0.0002$ which corresponds to the best baseline policy (on task reward) that we could achieve with tuning this hyperparameter manually. We note that task performance across all three policies is similar in simulation.

Results. Across both standing and walking tasks shown in Tab. 1, we find policies trained with PEGrad lead to significantly lower currents being drawn from the battery as well as reduced torques being applied compared to the factory policy. When deployed for the task of walking, the AMP+PPO $\lambda = 0$ policy was unsuccessful in completing the 12 feet distance and showcased unsafe behaviors such as taking large jumps – because of which we decided to omit it from the evaluation (marked with ✗ in the table). Interestingly, we find that the tuned torque penalty (AMP+PPO $\lambda = 0.0002$) and PEGrad achieve similar performance on all metrics for the standing task; however, we observe a *significant* gap in performance between the two on the walking task. PEGrad is 19.74% more efficient on current usage and 15.8% on net torques applied. We would like to point out that these numbers denote *per-timestep* current and torque usage and can show significant difference especially when a robot operates for a longer period of time.

Time Analysis. On a stand-alone RTX 4090 GPU, both PEGrad and the baseline AMP+PPO take ~4.5 hours to complete 10k iterations, achieving ~62k steps per second (SPS). On an HPC cluster with an L40 GPU, PEGrad takes 8.4 hours (~35k SPS), while the baseline takes 7.6 hours (~37k SPS). This difference likely stems from other bottlenecks such as simulation overhead rather than policy update, particularly given that most RL control policies use relatively shallow networks.

5 Conclusion

In this work, we introduced PEGrad, a method for incorporating energy minimization into RL-based robot control without compromising task performance. By projecting energy gradients orthogonal to task reward gradients, PEGrad avoids the need for sensitive hyperparameter tuning between the two objectives and prioritizes task success over energy objectives. Our extensive evaluations across DMControl, HumanoidBench, and real-world deployments on the Unitree Go2 quadruped demonstrate that PEGrad consistently achieves significant reductions in energy usage—up to 64% in simulation—while retaining or improving policy performance. Moreover, PEGrad enables Sim2Real transfer of energy-efficient policies, offering practical gains in battery usage and robot longevity.

6 Limitations

Interplay Between PEGrad and Style Rewards. Throughout this project, we found it challenging to combine the energy minimization objective with *style rewards*. In tasks like DMControl and HumanoidBench, which lack explicit style constraints, PEGrad effectively learned energy-efficient and task-performant policies. However, when we initially conducted experiments without AMP-based style rewards i.e traditional reward engineering with style-enforcing rewards such as *feet-air time* [8], *mirror loss* [42], and other penalties not directly tied to energy—e.g., *joint deviation*, *base acceleration*, and *action rate*—we observed interesting behaviors that were sometimes infeasible or unsafe for Sim2Real transfer. For instance, in the absence of a constraint on base height, one policy learned to maximize its base height as a way to reduce per-step torque, which resulted in a constrained set of allowable joint displacements. This, in turn, led to shorter, rapid steps so as to maintain the task reward. Another policy that was trained without any style reward robot learned a dragging behavior where 2 legs are actively moving and the rest are dragging themselves, minimizing the overall torque. Additional cases are discussed in Appendix 8.6. These instances depict the challenges while training with multiple reward functions and one potential way to alleviate this could be to recursively use PEGrad in the order of priority of objectives, with each of the objectives having a separate critic. We leave this as a future direction of our work.

7 Acknowledgements

SP would like to thank Mohit Gadde and Aayam Shrestha for their help with the real-world experiments, Hunter Brown for his inputs on energy formulation, Rob Yelle for his support of GPU resources during the deadline, and ViRL lab members for their feedback on the draft of this work. This work is supported in part by NSF Award No. 2321851. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government, or any sponsor.

References

[1] Unitree Robotics. Go2 developer documentation, 2024. URL https://support.unitree.com/home/en/developer/about_Go2. Accessed: 2025-04-24.

[2] J. Hwangbo, J. Lee, A. Dosovitskiy, C. S. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. In *Science Robotics*, 2019.

[3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. In *Science Robotics*, 2020.

[4] X. Long et al. Robust terrain-adaptive locomotion for legged robots via h-infinity reinforcement learning. In *IEEE Robotics and Automation Letters*, 2023.

[5] Y. Yang, T. Zhang, E. Coumans, J. Tan, and B. Boots. Fast and efficient locomotion via learned gait transitions. *Proceedings of the Conference on Robot Learning*, 2022.

[6] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. In *Robotics: Science and Systems (RSS)*, 2021.

[7] J. Siekmann, Y. Godse, A. Fern, and J. W. Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. *arXiv preprint arXiv:2011.01387*, 2020.

[8] B. van Marum, A. Shrestha, H. Duan, P. Dugar, J. Dao, and A. Fern. Revisiting reward design and evaluation for robust humanoid standing and walking. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[9] Z. Fu, X. Cheng, and D. Pathak. Deep whole-body control: Learning a unified policy for manipulation and locomotion. In *Proceedings of The 6th Conference on Robot Learning*, Proceedings of Machine Learning Research, 2023.

[10] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[11] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 2020.

[12] C. Sferrazza, D.-M. Huang, X. Lin, Y. Lee, and P. Abbeel. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *arXiv preprint arXiv:2403.10506*, 2024.

[13] W. Chen, X. Zhang, B. Lin, X. Lin, H. Zhao, Q. Zhang, and J. T. Kwok. Gradient-based multi-objective deep learning: Algorithms, theories, applications, and beyond. *arXiv preprint arXiv:2501.10945*, 2025.

[14] O. Sener and V. Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

[15] J.-A. Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350:313–318, 2012. URL https://api.semanticscholar.org/CorpusID:120459561.

[16] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, 2020.

[17] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 34:18878–18890, 2021.

[18] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR, 2018.

[19] Y. Du, W. M. Czarnecki, S. M. Jayakumar, M. Farajtabar, R. Pascanu, and B. Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

[20] Z. Chen, J. Ngiam, Y. Huang, T. Luong, H. Kretzschmar, Y. Chai, and D. Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. *Advances in Neural Information Processing Systems*, 33:2039–2050, 2020.

[21] P. Xu, X. Shang, V. B. Zordan, and I. Karamouzas. Composite motion learning with task control. *ACM Transactions on Graphics (TOG)*, 2023.

[22] R. Nai, J. You, L. Cao, H. Cui, S. Zhang, H. Xu, and Y. Gao. Fine-tuning hard-to-simulate objectives for quadruped locomotion: A case study on total power saving. *arXiv preprint arXiv:2502.10956*, 2025.

[23] Z. Fu, A. Kumar, J. Malik, and D. Pathak. Minimizing energy consumption leads to the emergence of gaits in legged robots. *arXiv preprint arXiv:2111.01674*, 2021.

[24] S. Mahankali, C.-C. Lee, G. B. Margolis, Z.-W. Hong, and P. Agrawal. Maximizing quadruped velocity by minimizing energy. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[25] C. F. Hayes, R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

[26] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. Technical report, MIT, 1999. Appears in NeurIPS 1999 workshop-style proceedings.

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012. doi:10.1109/IROS.2012.6386109.

[30] V. Moens and contributors. Leanrl: Turbo-implementations of cleanrl scripts. https://github.com/pytorch-labs/leanrl, 2024. URL https://github.com/pytorch-labs/leanrl. GitHub repository. Fork of CleanRL, optimized for PyTorch 2.0 features.

[31] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

[32] W.-C. Tseng. Weichengtseng/pytorch-pcgrad, 2020. URL https://github.com/WeiChengTseng/Pytorch-PCGrad.git.

[33] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi:10.1109/LRA.2023.3270034.

[34] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning*, Proceedings of Machine Learning Research. PMLR, 2022. URL https://proceedings.mlr.press/v164/rudin22a.html.

[35] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa. Amp: adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics*, 40(4):1–20, July 2021. ISSN 1557-7368. doi:10.1145/3450626.3459670. URL http://dx.doi.org/10.1145/3450626.3459670.

[36] Y. Wang, Z. Jiang, and J. Chen. Learning robust, agile, natural legged locomotion skills in the wild. *arXiv preprint arXiv:2304.10888*, 2023.

[37] H. Huang, W. Cui, T. Zhang, S. Li, J. Han, B. Qin, T. Zhang, L. Zheng, Z. Tang, C. Hu, et al. Think on your feet: Seamless transition between human-like locomotion in response to changing commands. *arXiv preprint arXiv:2502.18901*, 2025.

[38] A. Tang, T. Hiraoka, N. Hiraoka, F. Shi, K. Kawaharazuka, K. Kojima, K. Okada, and M. Inaba. Humanmimic: Learning natural locomotion and transitions for humanoid robot via wasserstein adversarial imitation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[39] A. Escontrela, X. B. Peng, W. Yu, T. Zhang, A. Iscen, K. Goldberg, and P. Abbeel. Adversarial motion priors make good substitutes for complex reward functions. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[40] F. Zargarbashi, J. Cheng, D. Kang, R. Sumner, and S. Coros. Robotkeyframing: Learning locomotion with high-level objectives via mixture of dense and sparse rewards. *arXiv preprint arXiv:2407.11562*, 2024.

[41] T. Li, Y. Zhang, C. Zhang, Q. Zhu, J. Sheng, W. Chi, C. Zhou, and L. Han. Learning terrain-adaptive locomotion with agile behaviors by imitating animals. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[42] M. Mittal, N. Rudin, V. Klemm, A. Allshire, and M. Hutter. Symmetry considerations for learning task symmetric robot policies. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7433–7439, 2024. doi:10.1109/ICRA57147.2024.10611493.

[43] H. Lee, D. Hwang, D. Kim, H. Kim, J. J. Tai, K. Subramanian, P. R. Wurman, J. Choo, P. Stone, and T. Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2025. URL `https://openreview.net/forum?id=jXLiDKsuDo`.

8 Appendix

8.1 Multi-Objective PPO

Proximal Policy Optimization (PPO). PPO [28] is a widely used on-policy algorithm that optimizes a surrogate objective based on the clipped probability ratio between the new and old policies. Specifically, PPO seeks to maximize the expected advantage while limiting the deviation from the previous policy using a clipped objective:

$$\mathcal{L}_{\text{PPO}} = \mathbb{E}_t \left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right] \tag{8}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the current and old policies, ε is a hyperparameter controlling the clip range, and $\hat{A}_t$ is an estimate of the advantage function. The value function is learned via regression to the discounted returns, and the policy and value networks are trained simultaneously. An entropy bonus is often added to the objective to encourage exploration.

Multi-Objective PPO. In our multi-objective setting, we extend PPO to optimize over both task performance and energy consumption. We maintain two separate reward signals: one for task reward r^r and another for energy usage r^e, and compute separate advantage estimates $\hat{A}_t^r$ and $\hat{A}_t^e$ accordingly. To combine them, we use a linear scalarization approach with a trade-off parameter λ, forming a scalarized advantage:

$$\hat{A}_t^{\text{total}} = \hat{A}_t^r - \lambda \hat{A}_t^e \tag{9}$$

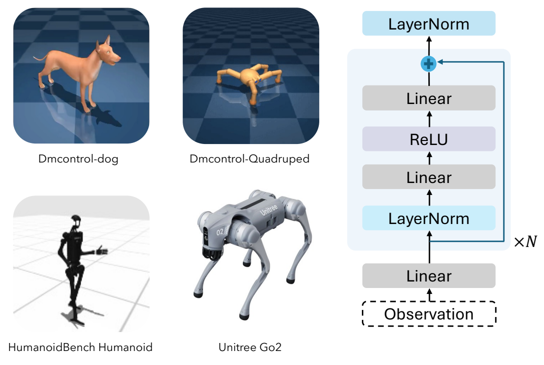
This scalarized advantage is used in the PPO surrogate objective:

$$\mathcal{L}_{\text{MO-PPO}} = \mathbb{E}_t \left[\min\left(r_t(\theta)\hat{A}_t^{\text{total}}, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t^{\text{total}}\right)\right] \tag{10}$$

As with SAC, setting the trade-off parameter λ is crucial: larger values more strongly penalize energy usage but can degrade task performance or convergence.

8.2 Environments

We consider 10 simulation tasks (six from `DMControl` suite [11] and 4 locomotion tasks from `HumanoidBench` benchmark [12] as shown in Figure 4. The dimensionality of the observation space and action space is mentioned in Table 2.



(a) Environments (b) Policy Encoder

Figure 4: *a) Environments* – We consider 4 locomotion environments – (i) `DMControl-Dog`, (ii) `DMControl-Quadruped`, (iii) `HumanoidBench-H1 Humanoid`, and (iv) `Unitree Go2`. We test `PEGrad` on 10 simulated tasks with `DMControl` and `HumanoidBench` environments and conduct a Sim2Real evaluation of standing and walking tasks with Unitree Go2 quadruped. *b) Policy Encoder* – We choose a SimBa-like [43] architecture that has shown sample-efficiency and better task performance by carefully selecting the architecture encoder for Actor and Critic in Deep-RL algorithms.

Task	**Observation dim**	**Action dim**
`quadruped-walk`	78	12
`quadruped-run`	78	12
`dog-stand`	223	38
`dog-walk`	223	38
`dog-trot`	223	38
`dog-run`	223	38
`h1-stand`	51	19
`h1-sit_simple`	51	19
`h1-walk`	51	19
`h1-run`	51	19
`Go2-Sim2Real`	48	12

Table 2: State and Action spaces of each of the tasks.

8.3 AMP + PPO Training

Dataset Collection. Demonstration trajectories are collected using a factory-provided controller by manually operating the robot via a joystick for the locomotion task, capturing joint positions and joint velocities at 50 Hz. This yields a 24-dimensional observation vector (joint positions and velocities) corresponding to 12 actuated joints. The trajectory spans 20 seconds, and we further augment the dataset by extracting all possible sliding windows of 0.2 seconds to get a large batch of trajectories.

The discriminator is implemented as a fully connected multilayer perception (MLP) with two hidden layers of 1024 and 256 units, respectively, with ELU activations. It receives the 24-dimensional proprioception as input and outputs a single logit that denotes the confidence that the original observation is from real demonstrations. The discriminator is trained jointly with the policy, with updates occurring at every learning step.

The style reward is derived by applying a sigmoid activation to the discriminator's logits to obtain the probability that a sample is classified as real. Denoting this probability by $p = \sigma(\text{logits})$, the style reward is then computed as the negative log-probability of the discriminator predicting "real":

$$r_{style} = -\log\left(\max\left(1 - \sigma(\text{logits}), \varepsilon\right)\right), \tag{11}$$

where ε is a small constant to prevent numerical instability. This reward encourages the policy to provide behaviors the indistinguishable from real demonstration. The style reward is scaled by a factor λ_{style} and combined with the task reward and penalties.

The task reward promotes velocity tracking locomotion by minimizing the L2 deviation between commanded and actual linear (X, Y) and angular velocities. Additionally, for the Baseline (`AMP+PPO`), we incorporate auxiliary penalty terms such as linear motion in the Z-axis and angular velocities about the X and Y axes (roll and pitch rates) to promote stable locomotion. Termination conditions, such as minimum base height and base contact, are also added. We describe the exact reward structure used to train GO2 SaW (Standing and Walking) controller with Adverserial Motion Priors (AMP) [35] in Table 3.

8.4 SAC Hyperparameters and Architecture

We show our encoder architecture for policy as well as for the critic in Figure 8.2 where we base our model on SimBa [43]. In this section we list out the hyperparameters for our SAC algorithm used on `DMControl` and `HumanoidBench` training in Table 4.

Table 3: Individual task, style and penalty rewards for `AMP+PPO` baseline.

Reward Term	**Definition**	**Weighting**
Base linear x, y velocity	$e^{-\frac{\|\mathbf{v}_{cmd}-\mathbf{v}_{current}\|^2}{0.25}}$	1.5
Base angular yaw velocity	$e^{-\frac{\|w_{cmd}-w_{current}\|^2}{0.25}}$	0.75
AMP Style reward	$-\log\left(\max\left(1-\sigma(\text{logits}), \varepsilon\right)\right)$	0.4
Feet air time	$\begin{cases} \sum_{k\in\mathscr{F}} (t_{\text{air},k} - t_{\text{thresh}}) \cdot \mathbf{1}_{\text{fc},k} & \text{if } \|\mathbf{v}_{\text{cmd}}\| > \delta \\ 0 & \text{otherwise} \end{cases}$	0.25
Vertical velocity penalty	$\|v_z\|^2$	-2.0
Roll/pitch velocity penalty	$\|w_{xy}\|^2$	-0.05
Orientation penalty	$\|\theta_{\text{rp}}\|^2$	-2.5
Torque penalty (not used in `PEGrad`)	$\|\tau\|^2$	-0.0002

Notes.

- $\mathbf{v}_{\text{cmd}}$ is the commanded linear velocity.
- $\mathbf{v}_{\text{current}}$ is the base linear velocity.
- $\mathbf{w}_{\text{cmd}}$ is the commanded angular velocity.
- $\mathbf{w}_{\text{cmd}}$ is the base yaw angular velocity.
- **logits** is a discriminator network output.
- ε is a numerical stability term to avoid computing log of zero.
- σ is the sigmoid function that computes the probability from the discriminator **logits**.
- $\mathscr{F}$ denotes the set of feet. In this case, it denotes four legs: front-left, front-right, rear-left, and rear-right.
- $t_{\text{air},k}$ is the air-time duration for foot k.
- t_{thresh} is the threshold air-time for the penalty.
- $\mathbf{1}_{\text{fc},k}$ is 1 if foot k just made contact with the ground, 0 otherwise.
- θ_{rp} is the roll and pitch angles of the robot base.
- δ is the minimum movement threshold to enable feet-air-time reward. This is required to disable the feet-air-time reward when the command is near zero.
- τ is the applied torque for each of the joints

8.5 Energy Formulation

In addition to formulating energy as sum of absolute joint torques, we consider mechanical power $(\tau.\omega)$. It is important to note that mechanical power can underestimate the current I drawn from the battery during high-torque, low-motion phases.

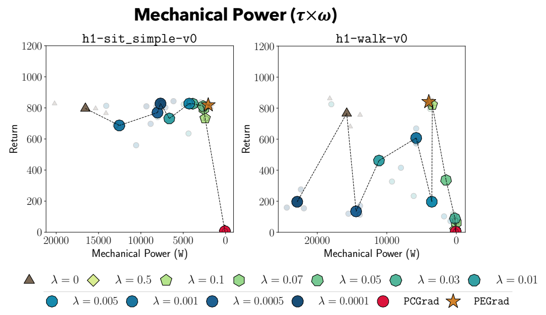


Figure 5: We show results on HumanoidBench's `sit_simple` and `walk` tasks with mechanical power $(\tau.\omega)$ as the energy formulation. We observe that `PEGrad` continues to achieve energy-efficient and task-performant policies even with a different formulation.

Table 4: **Hyperparameters Table.** The hyperparameters listed below are used consistently across all baseline and lagrange versions of SAC, unless stated otherwise.

	Hyperparameter	**Notation**	**Value**
Common	Discount factor	γ	0.99
	Replay buffer capacity	-	1M
	Buffer sampling	-	Uniform
	Batch size	-	256
Actor	Number of SimBa-like blocks	-	1
	Hidden dimension	-	{256/512} (`DMControl` / `HumanoidBench`)
	Entropy coefficient	α	0.2
	Update frequency	-	2
Critic	Number of SimBa-like blocks	-	2
	Hidden dimension	-	{256/512} (`DMControl` / `HumanoidBench`
	Update frequency	-	1
Optimizer	Optimizer	-	Adam
	Optimizer momentum	(β_1, β_2)	(0.9, 0.999)
	Weight Decay	-	0.0
	Policy LR	-	3e-4
	Critic LR	-	1e-3

8.6 Sim2Real Discussion

In this section, we first show some additional quantitative results that we conducted on Unitree Go2 robot for Standing and Walking (SaW) tasks. Next we highlight some additional examples of challenges we encountered during the process of performing Sim2Real experiments with style rewards.

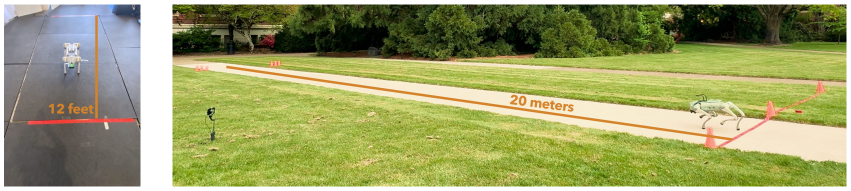


Figure 6: **Real world setup.** *(left)* Go2 in Lab-setting: We consider a rubber mat flooring terrain on which the dataset for AMP has been collected. For the results in Table 1, we consider a distance of 12 feet ($\sim$ 3.66 meters) (marked in red) where the robot is commanded a velocity of 0.5 m/s. *Right* Outdoor setting where we test our policies on the `concrete` pathway as well as the adjacent `grass` terrain. For this, we consider a larger distance of 20 meters (distance between the cones).

Additional Sim2Real results.

We deployed the trained SaW policies on different terrains to see if the lower energy consumption that was trained using reference trajectories that were collected in a lab-setting in Section 4.2, transferred similarly to outdoor terrains such as grass and concrete. All the trained Go2 policies were trained with dynamics randomization on a *flat terrain*. It is important to note that these set of experiments are *not* to evaluate the generalization capability of the policy to various terrains – rather to see if there is any significant deviation in terms of energy consumption of `PEGrad` versus the baselines. We report our results in Table 5.

Across both the scenarios, we find `PEGrad` to perform significantly better in terms of both current drawn as well as the net torque applied. Specifically, we find `PEGrad` to be better than the finetuned `AMP+PPO` $\lambda = 0.0002$ baseline by $\sim$ 24.5% on concrete and $\sim$ 7.68% on grass. We suspect this reduction of energy consumption even for the baseline on grass terrain is because grass can absorb part of the impact, reducing the need for active damping or stabilization from the motors. On the other hand, the average current drawn on `Concrete` is similar to the lab-setting where the robot was deployed on a rubber sheet. One important observation was that in grassy & concrete terrains the policy sometimes ends up at a lower speed than commanded. We attribute this sim2real gap to the

SaW Controller	`Walking (Concrete)`		`Walking (Grass)`	
	Current Drawn (mA)	Net Torque Applied (Nm)	Current Drawn (mA)	Net Torque Applied (Nm)
Factory	5.430 ±0.007	3.97 ±0.009	4.875 ±0.181	3.72 ±0.050
`AMP+PPO` $\lambda = 0.0002$	6.905 ±0.753	4.47 ±0.115	4.669 ±0.384	3.71 ±0.077
AMP+PPO `PEGrad`	**5.208 ±0.399**	**3.45 ±0.014**	**4.310 ±0.002**	**3.26 ±0.005**

Table 5: Current and Torque usage on different terrains: We compare Unitree's `Factory` controller and `AMP+PPO` λ baselines against `PEGrad` for `Walking` task on `Concrete` and `Grass` and report current drawn and net torque applied. We find that `PEGrad` is ~24.5% more efficient than the tuned multi-objective `AMP+PPO` λ=0.0002 on `Concrete` terrain and ~7.68% on `Grass` terrain. Results averaged over two trials.

fact that all our GO2 policies were trained on flat terrain. We hypothesize that further training of policies on a diverse terrain would alleviate this issue.

Additional examples for interplay between `PEGrad` and Style Rewards

As discussed in our limitations (Section 6), one of the challenges we faced with minimizing energy was to learn a behavior alongside style rewards. In addition to the examples mentioned in the Limitations section, we often observed hopping or trotting behaviors in simulation when working with different style reward structures. With a reward structure the adds a mirror loss [8] which is common in several humanoid and quadruped locomotion literature to encourage the gaits to be symmetric about the sagittal plane, the robot learns a hopping behavior (we show a video of this in the supplementary video).