ANALYTIC POLICY GRADIENTS FOR CONTACT-RICH LOCO-MANIPULATION IN SOFT ROBOTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Analytic policy gradients are widely considered impractical for contact-rich manipulation, where contact discontinuities induce bias and stiff or chaotic dynamics can produce high-variance estimates and exploding gradients over long horizons. Contrary to this view, we demonstrate contact-rich, soccer-inspired loco-manipulation in soft robots with up to 2,420 actuated springs, trained end-to-end with first-order gradients through simulation and control. On one commodity GPU, a single agent learned to locate, pursue, dribble, and kick the ball through the goal in 100 gradient steps—less than a minute of wall-clock time. In multi-agent settings, cooperative "passing" emerged strictly from task loss without explicit behavioral shaping, and in a competitive setting a defender—tasked with preventing goals—reduced its opponents' scoring rate by up to 60%. Gradient analysis reveals that sustained contact events inflate gradient energy by up to two orders of magnitude and leave per-window gradients largely misaligned, yet batch aggregation yields a reliable training signal. We also find that gradient energy is skewed away from the wide output layer that drives spring actuation, starving it of the sparse contact-mediated learning signal under plain stochastic gradient descent (SGD). With per-layer rescaling, SGD nearly matches Adam without per-coordinate adaptation. We release our simulator and task environment as a scalable, fully differentiable foundation for contact-rich, multi-agent learning.

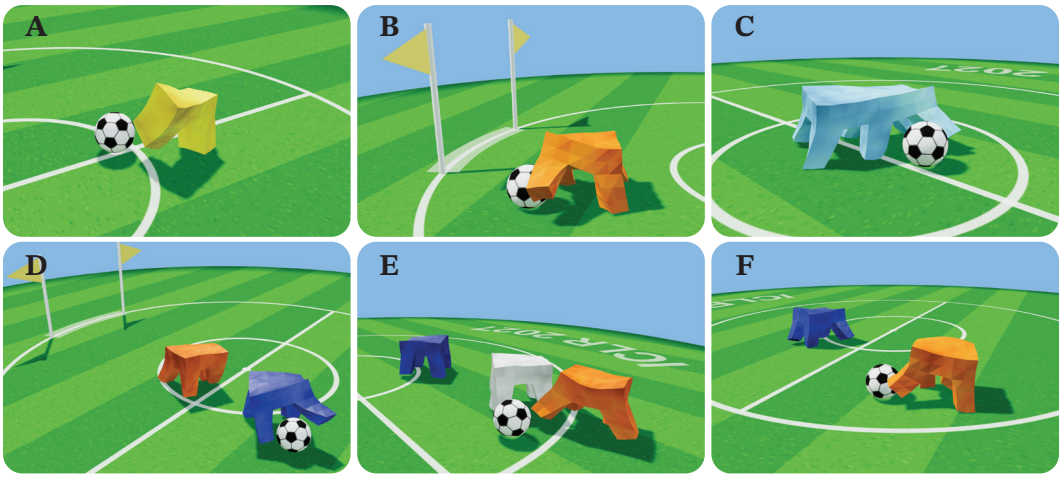

Figure 1: **Soft robot soccer.** A biped (**A**), quadruped (**B**), and hexapod (**C**) learned to approach, dribble, kick, and score in 100 analytic policy gradient updates. A cooperative pair (**D**, **F**) trained with a shared policy and objective demonstrated emergent "passing" to transport the ball to the goal. A defender (white body, **E**) learned to guard the ball near the boundary to obstruct transport.

1 INTRODUCTION

Soft robots replace rigid links and joints with compliant bodies, so that behavior arises from continuum deformation rather than from rotation about a fixed set of axes (Kim et al., 2013). That

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.

such compliant structures can do useful work has ample precedent in the natural world: elephant trunks, octopus arms, and caterpillar bodies exhibit dexterous manipulation and adaptive locomotion through actuation of soft tissue alone (Kier & Smith, 1985). The promise of realizing these capabilities in machines is of interest practically—compliance yields safe human–robot interaction, robustness to impact, and conformal grasping (Cianchetti et al., 2018)—and scientifically, because soft bodies pose a distinct control problem (Rus & Tolley, 2015). With many degrees of freedom (DoFs) and low mechanical impedance, a continuum body admits no general-purpose low-dimensional action space that preserves its compliant repertoire (Pfeifer et al., 2012).

Optimizing policies with large action spaces is difficult for model-free reinforcement learning. Yet reinforcement learning is the dominant paradigm for controlling rigid-body robots, where, by contrast to soft robots, joints form a native low-dimensional action space. Because the variance of trial-and-error (zeroth-order) gradient estimation scales with the action dimension (Suh et al., 2022; Mohamed et al., 2020), these methods are tractable at tens of DoFs but demonstrably strained beyond that (Xu et al., 2021b; Schumacher et al., 2023). As a result, prior work in large action spaces has engineered the problem back down to size by, for example, performing unsupervised learning of coherence across actuators to guide exploration (Schumacher et al., 2023) or inserting a low-dimensional interface between policy and actuators (Lee et al., 2019). Soft-robot control has done the same by reducing behavior to low-dimensional, open-loop policies (Cheney et al., 2013) and confining bodies to simple, small, often 2D morphologies (Bhatia et al., 2021).

Rather than engineering the morphology or action space to accommodate the sample complexity of reinforcement learning, an alternative approach to policy optimization is via the first-order gradients—often referred to as analytic policy gradients—exposed by a differentiable simulator (Hu et al., 2019a;b; Freeman et al., 2021). These are exact gradients of the objective, as in supervised learning, and their variance does not scale with the action dimension (Suh et al., 2022; Mohamed et al., 2020). Analytic policy gradients have been used to train closed-loop locomotion policies for rigid (Xu et al., 2021b; Georgiev et al., 2024; Xing et al., 2025) and soft (Ma et al., 2021; Strgar et al., 2024; Sato et al., 2025) bodies, and manipulation policies for rigid (Xu et al., 2021a; Huang et al., 2021; Xing et al., 2025) and soft (Wang et al., 2023; Qiao et al., 2021) bodied effectors. Multi-agent systems have been optimized through differentiable simulators, but agents were either modeled as contact-free point masses (Zhang et al., 2025) or devoid of dynamics altogether (Kuroki et al., 2023).

Here we demonstrate for the first time contact-rich loco-manipulation in soft robots trained with analytic policy gradients. In a soccer-inspired task, legged robots—a biped, quadruped, and hexapod—learned to locate, pursue, dribble, and kick the ball through the goal[1] in 100 gradient steps—less than a minute of wall-clock time on a commodity GPU. In multi-agent settings, cooperating agents learned "passing" from the task objective alone, without behavioral shaping, and in a competitive setting a defender trained to prevent goals reduced its opponents' scoring rate by 60%. To the best of our knowledge, this represents the first instance of multi-agent learning in a differentiable, contact-rich environment.

Our method is deliberately minimal: egocentric observations are mapped to actions by a deterministic policy (Sect. 2.3), which is optimized by backpropagating the task loss through 500-step windows of differentiable physics (Sect. 2.4) over long-horizon episode rollouts (Fig. 3). The closest prior work, SHAC (Xu et al., 2021b), trained rigid-body locomotion with analytic policy gradients truncated at 32-step window boundaries and depended on a learned critic—without it, no task was solved. Despite the added complexity of our setting—soft bodies in place of rigid, an order of magnitude larger action space, and an extension to contact-rich loco-manipulation—we found the critic unnecessary, training with $20\times$ fewer policy updates and, without the critic's $> 10^5$ updates, three orders of magnitude fewer gradient steps in total.

That bare analytic policy gradients suffice for closed-loop, contact-rich loco-manipulation over long-horizon episodes is surprising—they are widely considered impractical in this setting. Suh et al. (2022) catalogue several reasons including estimator bias induced by contact discontinuities and variance from stiff or chaotic dynamics that can compound into exploding gradients over long horizons (Metz et al., 2021). Antonova et al. (2023) found, in a setting similar to ours—a mass-spring deformable body in dynamic contact with a rigid object—that gradients were frequently incorrect,

[1]Project page with videos: https://phenomenal-banoffee-9015bf.netlify.app

despite forward simulation appearing realistic. Newbury et al. (2024) summarizes the resulting practice: successful uses of analytic policy gradients have typically been confined to fixed initial states, open-loop control, short horizons, or simplified contact models. In more complex settings they often require augmentation: a learned critic (Xu et al., 2021b), noise injected to smooth the objective or a blend of first- and zeroth-order estimates (Suh et al., 2022), global search in lieu of descent (Antonova et al., 2023), or gradients used only to generate trajectories that a policy later imitates (Mora et al., 2021).

We characterized the gradients underlying our results and found certain predicted pathologies present but tractable. Under sustained robot–ball contact, gradient energy rose by up to two orders of magnitude, and contact gradients exhibited near-zero pairwise alignment; nonetheless, aggregation over large batches yielded a reliable training signal. We also observed an imbalanced gradient allocation, concentrated away from the wide output layer driving robot actuation. Stochastic gradient descent, with a single learning rate, followed this allocation and performed poorly, while Adam and SGD with per-layer rescaling redistributed update energy across layers and achieved similar, superior results. Our results provide a proof-of-concept for analytic gradients in contact-rich, multi-agent loco-manipulation. We release our simulator and task environment as a platform for future work.

2 METHODS

We trained neural control for soft robot loco-manipulation with analytic gradients through a differentiable simulator. Section 2.1 defines the task, Sect. 2.2 the soft robots, and Sect. 2.3 the neural controllers that drove them. Section 2.4 presents the differentiable simulator that computed the environment's dynamics. Sections 2.5–2.8 describe training: the episode distribution, the objective, gradient computation through simulation rollouts, and optimization.

2.1 TASK

Robots were tasked with transporting—kicking, dribbling, passing—a ball toward and through a goal, a soccer-inspired formulation of loco-manipulation. We investigated this task in three settings: (1) a **single agent**; (2) a **cooperative** pair; and (3) a **competitive** setting in which the pair faced an opposing defender, charged with preventing goals. All three were instances of one environment with the field, ball, physics, and rules held constant. The field was an origin-centered circle of radius $R = 3.5$ m (Fig. 2) marked on flat ground. The goal was a 1.75 m arc on the boundary line ($\approx 8\%$ of the field's circumference). A goal was scored when the ball's center of mass crossed the boundary within the goal arc. Other ball–boundary crossings incurred a penalty, but the ball remained in play. Robots could cross the boundary freely.

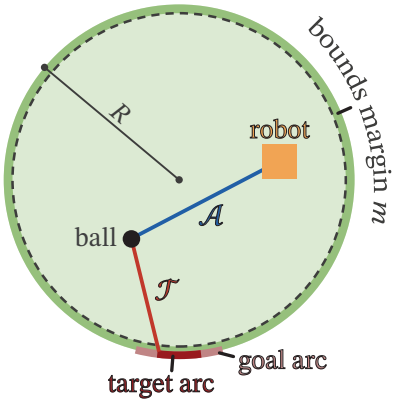

Figure 2: **Task geometry.** Robots played a soccer-inspired loco-manipulation game on a circular field ($R = 3.5$ m), scoring by transporting the ball through a 1.75 m "goal" arc. Task objectives (Sect. 2.6) included minimizing the robot–ball distance $\mathcal{A}$ and the ball's distance $\mathcal{T}$ to the "target" arc, the goal's inner 50%. A boundary penalty applied when the ball was within margin m of the field's edge, or beyond it. Drawn to scale.

2.2 SOFT ROBOTS

Robots were built from soft, actuated voxels modeled as mass-spring networks (Fig. 6) (Hiller & Lipson, 2012; Cheney et al., 2018; Kriegman et al., 2020; Strgar & Kriegman, 2026). Each voxel was a 0.1 m cube: point masses were positioned at the eight corners and springs connected masses along cube edges and face diagonals. Voxels were arranged contiguously to form legged robot morphologies, namely a biped, quadruped, and hexapod, with adjacent voxels sharing incident masses and springs. In multi-agent settings, all robots shared the quadruped morphology. Legs were sheared outward and tapered to avoid self-collision during locomotion. We refer to the resulting configuration as the robot's *rest pose*—each spring's free length was initialized to the distance $\bar{\ell}_0$ between its endpoint masses so the mass-spring network was stress-free by construction.

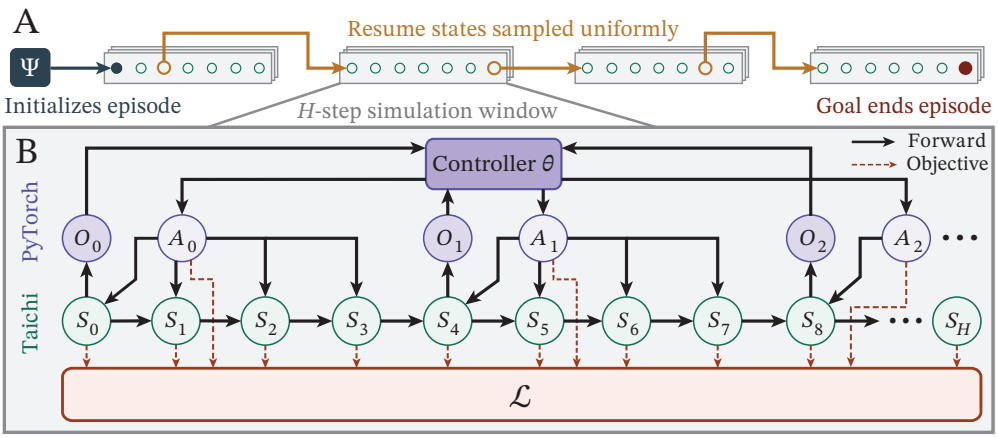

Figure 3: **Training episode rollout.** (**A**) A batch of B episodes was initialized from prior distribution Ψ and unrolled across overlapping H-step simulation windows. Noninitial windows began from simulation states sampled uniformly from predecessor windows, and episodes terminated when a goal was scored. (**B**) Each window advanced from its initial state S_0 by interleaving forward simulation with calls to the PyTorch-based controller. Every $n = 4$ steps, the controller, with parameters θ, mapped observation O_i from the Taichi-based simulator's state to action A_i, held constant for the next n simulation steps. The complete forward rollout—spanning states, observations, actions, and parameters—was end-to-end differentiable. State and action terms accumulated into the window objective $\mathcal{L}$, and analytic gradients $\partial\mathcal{L}/\partial\theta$ were computed by reverse-mode automatic differentiation.

2.3 Neural Control

Observations were egocentric: the robot perceived only relations among itself and the task objects—ball, goal, field center, and, when present, other robots. Thirty-two sensors were positioned by farthest point sampling (Eldar et al., 1997) at masses across a robot's body (Fig. 6), and each measured its distance to task objects. Readings were grouped by object and mean-centered within each group, discarding range and yielding a body-frame directional layout of the scene. Ranges entered separately as center-of-mass distances among task objects and the robot itself, together with scalar rates such as the robot's closing rate toward the ball and the ball's toward the goal. Finally, 10 sinusoidal clock signals (≈ 2.4 Hz; equally spaced phases) provided a gait-timing reference (Ijspeert, 2008). Every input was invariant to rotations and translations of the world frame.

Observations were mapped to actions by a multi-layer perceptron with two hidden layers of width 128, each followed by ReLU activation and layer normalization. A $\tanh$ output layer produced one action, $a \in [-1, 1]$, per spring (Sect. 2.2), modulating its free length in the simulator (Sect. 2.4). Cooperating robots shared a single network, with each robot's clock phases shifted to break symmetry. Across morphologies and task settings, networks comprised up to 343,796 parameters, hereafter denoted θ. See Appendix A.1.2 for additional detail.

2.4 Differentiable Simulator

The differentiable simulator was adapted from Strgar & Kriegman (2026) to add the dynamics of a rigid spherical body (the manipulated ball), frictional contact between robots and the ball, and robot–robot collisions.

Robots were simulated as point masses connected by springs (Sect. 2.2), and the ball as a rigid, solid sphere with a radius of 17.5 cm. Each spring exerted a Hookean restoring force of magnitude $f = k_s(\ell - \ell_0(a))$ on its endpoint masses, along the spring axis. Here, k_s is the spring constant, ℓ is the spring's actual length, and $\ell_0(a) = \bar{\ell}_0(1 + 0.065a)$ its free length, modulated about the initial rest-pose length $\bar{\ell}_0$ by the controller's action $a \in [-1, 1]$ (Sect. 2.3). The controller was invoked once every $n = 4$ simulation steps (≈ 83 Hz), and actions were held constant between calls. Simulation states were advanced by semi-implicit Euler integration with $\Delta t = 3$ ms under gravity with viscous damping of all velocities.

Robot and ball collisions with the flat ground plane were resolved with continuous time-of-impact handling, with restitution and Coulomb friction. Robot–ball contact was penalty-based: interpenetration produced a normal spring–damper force and a Coulomb friction force opposing relative tangential motion. Collisions produced torque on the ball and friction at the contact point, so the ball could roll, spin, and be kicked and dribbled. In multi-agent settings, robot–robot contact was modeled as a frictionless spring–damper acting between cross-robot mass pairs. Self-collision within a robot's body was not modeled.

The simulator was written in the Taichi programming language (Hu et al., 2019a), providing GPU acceleration and analytic simulation gradients via reverse-mode automatic differentiation (Hu et al., 2020). Because observations and actions passed directly between simulation state and our PyTorch-based controller (Sect. 2.3), gradients flowed end-to-end through physics and control (Sect. 2.7). Our complete simulator, including parameter configurations, will be provided with the released code. Parameters were not calibrated to a physical system; reported units are nominal.

2.5 Episode distribution

Training episodes were sampled from a prior distribution, Ψ, over initial robot, ball, and goal positions. Robot positions were area-uniform within the field (Sect. 2.1). The ball, which always spawned at rest, was placed uniformly in angle and radial distance from the field center, biasing it away from the boundary. Finally, the goal arc position was uniform on the boundary. Samples were rejected and redrawn if any two robots, the ball and any robot, or the ball and goal fell within a minimum separation, chosen so that no robot spawned overlapping another and no episode began nearly solved.

Rollouts ran in a fixed batch of B parallel environments, or *slots*. Each slot was simulated in successive windows of $H = 500$ steps (1.5 s of simulation time; Sect. 2.4; Fig. 3A), with gradients truncated at window boundaries (Sect. 2.7). At the controller's clock frequency (Sect. 2.3), each window spanned ≈ 3.6 gait cycles. In the first training step, windows began with episodes drawn from Ψ, with robots placed in their rest pose (Sect. 2.2). In subsequent steps, each window began from a state sampled uniformly at random from its predecessor within the same slot. A slot thus advanced through a lineage of windows within a single episode.

A scored goal (Sect. 2.1) truncated the window, ending the episode and its lineage. Window lineages also terminated once reaching a depth of 27 windows (20 s of expected simulation time). This cap guaranteed a reliable inflow of fresh episodes and bounded the age of resumed states independent of policy performance. Terminated slots were refilled from Ψ, but in the single-agent setting, robots were not placed at rest. Instead, the initial pose and velocity were sampled from mid-gait states harvested from the preceding step's rollouts and translated to the new episode's spawn positions. Multi-agent settings always initialized from rest pose.

2.6 Objective

Consider a window of H simulation steps (Sect. 2.5). At step t, let $b_t \in \mathbb{R}^2$ denote the ball's planar position and $c_t^{(r)} \in \mathbb{R}^2$ the planar center of mass of robot r. Let t_G equal the first step at which the ball crossed the goal arc (Sect. 2.1) or H if no goal occurred. Because a goal truncated the window, no loss term accrued past t_G.

Let $a_t \in [-1, 1]^S$ denote the actions (Sect. 2.3) applied at window step t to the S springs across all goal-scoring robots (either a single agent or a cooperating pair). Define t_A as the step at which any goal-scoring robot first contacted the ball (Sect. 2.4) or H if no contact occurred.

The per-window goal-scoring loss was a sum of four terms, $\mathcal{L}_G = \mathcal{A}+\mathcal{T}+\mathcal{B}+\mathcal{E}$, and the optimized objective (Sect. 2.8) was the summed window loss over a batch of B rollouts.

Approach, $\mathcal{A} = 1/(H\bar{d}_{\mathcal{A}}) \sum_{t<t_A} \min_r \left\| c_t^{(r)} - b_t \right\|$, charged the nearest goal-scoring robot's distance to the ball at every step before first contact. The normalizing constant, $\bar{d}_{\mathcal{A}}$, is a Monte-Carlo estimate of the expected initial minimum robot–ball distance under Ψ (Sect. 2.5).

Transport, $\mathcal{T} = 1/(H\bar{d}_{\mathcal{T}}) \sum_{t\,:\,\|b_t\|<R} D_G(b_t)$, charged the ball's distance to the goal, summed over steps at which the ball was in play (inside the field radius R; Sect. 2.1). D_G is the distance to

the inner 50% of the goal arc (hereafter the target arc; Fig. 2). When the ball's bearing lies within the target arc, D_G is purely radial; otherwise, it is the distance to the nearest endpoint of the target. Analogous to $\bar{d}_{\mathcal{A}}$, $\bar{d}_{\mathcal{T}}$ is the expected initial D_G under Ψ.

In the single-agent setting the sum was replaced by the window's terminal value, $\mathcal{T} = D_G(b_{t_G})/\bar{d}_{\mathcal{T}}$. There, we found the summed form produced conservative dribbling or poorly aimed kicks, whereas charging only the terminal position left the trajectory unconstrained while still demanding accurate shots.

Bounds, $\mathcal{B} = 1/H \sum_t \rho(b_t)$ with $\rho(b) = \gamma \max\big(0,\ \|b\| - (R - m)\big)$, charged when the ball was near or beyond the field boundary (Sect. 2.1). The margin, m, is equal to the ball's radius, so the penalty was incurred when the ball's edge reached the boundary (excluding the goal's angular sector). The penalty slope was $\gamma = 0.4\ \mathrm{m}^{-1}$.

Effort, $\mathcal{E} = \lambda/(HS) \sum_t \|a_t\|^2$ with $\lambda = 0.05$, penalized actuation energy to encourage efficient loco-manipulation.

In the competitive setting, the defender network was trained with the per-window loss $\mathcal{L}_D = \mathcal{A}_D - \mathcal{T} + \mathcal{B} + \mathcal{E}_D$, where $\mathcal{A}_D$ and $\mathcal{E}_D$ are the approach and effort terms applied to the defender, and $\mathcal{T}$ takes the summed form.

2.7 Gradient computation

Let $\mathcal{L} \in \{\mathcal{L}_G, \mathcal{L}_D\}$ denote the objective (Sect. 2.6) and θ the corresponding controller parameters (Sect. 2.3). In each training window (Sect. 2.5), analytic gradients $\partial\mathcal{L}/\partial\theta$ were computed by back-propagation through the full H-step window. Gradients did not cross window boundaries, and no alternative mechanism—e.g., a terminal value function (Xu et al., 2021b)—was used to approximate gradients or assign credit beyond the window.

The PyTorch controller and Taichi simulator produced two separate computational graphs, each differentiated natively, with values and gradients exchanged on-GPU via Taichi kernels that read and wrote PyTorch tensors. During the forward pass, observations were passed from simulation state to the controller, which was evaluated, and its action written back into simulation state. The backward pass unwound both graphs in reverse, exchanging gradients at the same boundaries: each action received the sum of its gradients over the n simulation steps it governed (Sect. 2.4), and each observation accumulated its gradient back into the simulator state it was read from. Observations were therefore differentiable inputs, not detached sensor readings.

No stop-gradient operations were applied to simulator or controller states, and all classes of contact—robot–ball, robot–robot, robot–ground, and ball–ground—were included in the backward computation. Contact events were treated piecewise, with gradients computed in the locally active branch. The discrete branch-switching decision itself was not differentiated, and event times t_A and t_G, which masked the objective's accumulators (Sect. 2.6), were treated as constants during the backward pass.

2.8 Optimization

The controller (Sect. 2.3) was trained to minimize the per-window objective $\mathcal{L}$ (Sect. 2.7) summed over a batch of $B = 256$ parallel windows (Sect. 2.5). In the single-agent setting we trained for 100 gradient steps and compared four optimizers: Adam (Kingma & Ba, 2015); momentum-free stochastic gradient descent (SGD); SGD-norm, which rescaled each step to unit norm; and SGD-layer, which rescaled each parameter tensor's gradient so that its share of update energy matched its share of parameters. Multi-agent policies used Adam. The cooperative pair was trained for 600 steps. In the competitive setting, the pair was initialized from the pretrained cooperative policy and frozen while a defender was trained from scratch for 500 steps; the pair was then fine-tuned for 300 steps against the frozen defender. Learning rates followed a cosine schedule with 5% linear warmup and decay to 1% of the initial value, which was selected by sweep in the single-agent setting and reused elsewhere. The fine-tuned pair was the only exception—it used a 20% warmup and a manually tuned initial learning rate. Throughout, Adam used $\beta_1 = 0.85$, $\beta_2 = 0.95$ and L2 weight decay of 10^{-6} was applied. All simulation and optimization ran on one GeForce RTX 5090 GPU. See Appendix A.1.3 for additional details.

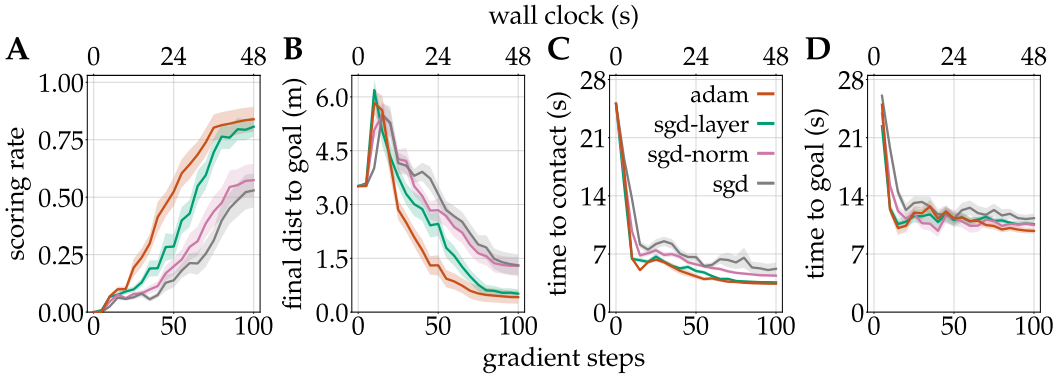

Figure 4: **Single-agent performance.** A quadruped was trained for 100 gradient steps with four optimizers and evaluated every fifth step on a fixed set of 500 episodes drawn from Ψ. Performance was measured as scoring rate (**A**), the ball's final distance to goal (**B**), time to first robot–ball contact over episodes with contact (**C**), and time to goal over scoring episodes (**D**) (mean $\pm$ standard error over 20 seeds). All optimizers learned locomotion nearly identically (C), but varied widely in scoring and manipulation (A, B). Top axes give wall-clock time on one RTX 5090.

3 Results

In this section we report single-agent loco-manipulation in a quadruped, comparing optimizers, and extensions to other morphologies (Sect. 3.1). We next examine the gradient profile underlying these results (Sect. 3.2). Finally, we evaluate extensions to multi-agent play (Sect. 3.3).

3.1 Single-agent loco-manipulation

Loco-manipulation performance of a quadruped was measured by drawing a fixed evaluation set of 500 episodes from Ψ (Sect. 2.5) with robots initialized at rest and a 30 s cap (non-limiting; Fig. 7). Every fifth gradient step, we measured scoring rate (the fraction of episodes ending in a goal), final ball–goal distance, time to first contact over episodes with contact, and time to goal over scoring episodes. In Figure 4, we report the mean of each measure ($\pm$ standard error) over 20 seeds—random initializations of the policy and episode sampler—for optimizers Adam, SGD-layer, SGD-norm, and SGD (Sect. 2.8). Locomotion was learned first, almost identically by all optimizers, settling to an ≈ 5 s average time to contact within 20 gradient steps (Fig. 4C). Manipulation followed with the final ball–goal distance rising initially, coincident with robots arriving at the ball and knocking it aimlessly, then falling with the optimizers settling into two groups: Adam with SGD-layer at ≈ 0.5 m and SGD-norm with vanilla SGD at ≈ 1.3 m (Fig. 4B). The scoring rate (Fig. 4A) separated along the same lines with Adam, SGD-layer, SGD-norm, SGD achieving final performance of 0.84 ± 0.05, 0.81 ± 0.04, 0.57 ± 0.07, 0.53 ± 0.07, respectively. Per-seed scoring-rate trajectories (Fig. 8) illustrate that the performance gap stems from reliability: the four optimizers, in the same order, scored above 80% in 16, 11, 5, and 5 of 20 seeds. Meanwhile, similar convergence in the time to goal measure (Fig. 4D) indicates that optimizers differed primarily in how often they scored, not how quickly. Each training run completed in 100 gradient steps or approximately 48 s of wall-clock time. A hexapod and a biped trained with Adam using the same hyperparameters also learned to score within 100 gradient steps, reaching 0.73 ± 0.13 and 0.92 ± 0.06 scoring rates, respectively (5 seeds; Fig. 10). Evaluation sets for the three morphologies were drawn independently from Ψ (Sect. 2.5), differing only in minimum separation constraints that scale with body size. In Figure 9A–E, we show the quadruped's behavioral trajectories, computed using the final checkpoint of one Adam run on five evaluation episodes.

3.2 Gradient analysis

Analyses in this section use the single-agent quadruped at $B = 256$ unless stated otherwise. Batch-gradient energy (Fig. 12) and each layer's share of it (Fig. 14) were logged in the training runs of Sect. 3.1. All other gradient statistics come from additional runs that recorded per-window gradients at every step: ten seeds for Adam and one per SGD variant.

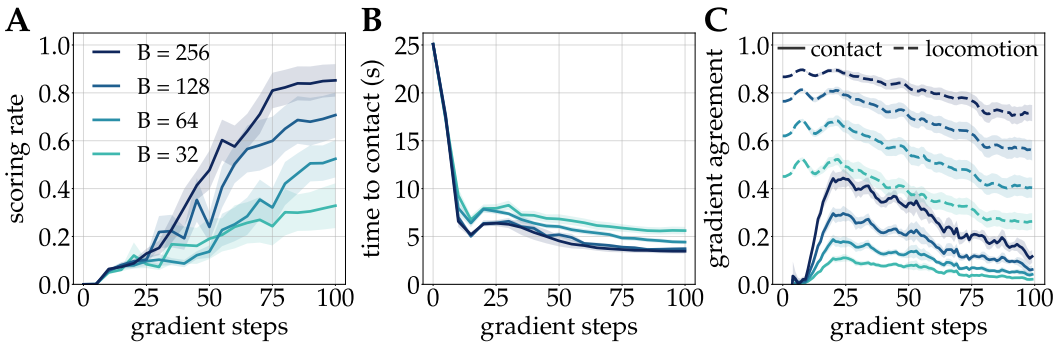

Figure 5: **Batch sweep.** Training with Adam at batch sizes $B \in \{32, 64, 128, 256\}$ over 10 seeds, each evaluated as in Section 3.1. Scoring rate (**A**) rose monotonically with B while locomotion (**B**) was learned at every B, suggesting the task performance gap reflects manipulation. (**C**) Gradient agreement—cosine similarity between the contact (solid) or locomotion (dashed) aggregates of two disjoint $B/2$-window subsets, drawn at each step of ten separate $B = 256$ runs to isolate the effect of batch size from variation in training trajectory. Aggregated locomotion gradients agreed substantially even at the smallest B; contact gradients agreed relatively weakly, and only when aggregated over a hundred or more windows. All curves show mean $\pm$ standard error over seeds. Those in C are smoothed with a 5-step centered moving average.

Contact gradients are large and incoherent. Per-window gradient energy—squared L2 norm—rose sharply with the fraction of window steps in which the robot touched the ball: windows in sustained contact ($> 5\%$ of steps) exhibited $\approx$ 10–80$\times$ the energy of no-contact windows across all optimizers (Fig. 11). Batch-aggregate gradient energy rose by 1.5–2.5 orders of magnitude early in training—coincident with the rise in contact frequency as locomotion was learned—and then plateaued (Fig. 12). Within a batch, gradients were largely misaligned—under Adam, the mean pairwise cosine similarity across all gradient steps was 0.02 overall, 0.09 among no-contact (locomotion) windows, 0.01 among contact windows, and -0.01 between locomotion and contact windows.

Batch aggregation recovers the manipulation signal. To test the role of aggregation in producing a usable update from poorly aligned per-window gradients, we trained Adam for 10 seeds at batch sizes $B \in \{32, 64, 128, 256\}$. Final scoring rate rose monotonically with batch size, from 0.33 ± 0.09 at $B = 32$ to 0.85 ± 0.07 at $B = 256$ (Fig. 5A), whereas locomotion was learned at every batch size, with relatively modest gain in time to contact from the smallest to the largest batch (Fig. 5B). To isolate the effect of aggregation from that of the training trajectory, we emulated each batch size using per-window gradients from the ten $B = 256$ runs: at every training step we drew two disjoint subsets of $B/2$ windows (for each B), aggregated the contact and no-contact windows within each subset separately, and measured the cosine similarity between the two subsets' aggregates of each type, which we term gradient agreement (Fig. 5C; mean $\pm$ standard error over runs). Locomotion agreement was substantial from the first step, even at $B = 32$ (mean over training: 0.39), and roughly doubled at $B = 256$ (0.81). Contact agreement rose from near zero as time to contact fell (Fig. 5B), peaked near step 25, and declined steadily thereafter. Averaged over training it was near zero at $B = 32$ (0.06), rose with batch size, but even at $B = 256$ (0.27) remained below the locomotion agreement of the smallest batch. Locomotion agreement also declined after step 25, but more gradually.

Gradient energy is skewed across layers. To probe the weaker performance of SGD and SGD-norm (Fig. 4), we measured each layer's share of batch gradient energy (Fig. 14). From initialization, energy was skewed relative to parameter count: the input layer held 26% of gradient energy with 6% of the parameters, and the output layer 38% with 86%—an order of magnitude less energy per parameter at the action head. SGD and SGD-norm, with one learning rate across the network, followed this allocation: SGD retained it throughout training, while SGD-norm's skew increased modestly. Under Adam the skew grew much further—the input layer's share reached $\approx 65\%$ and the output layer's $\approx 10\%$—but per-coordinate normalization made its updates indifferent to the raw allocation. This motivated SGD-layer, which, within each tensor, steps in the same direction as SGD-norm, rescaled to the tensor's parameter share, leaving the total step norm unchanged (Appx. A.1.3). This

reallocation alone raised final scoring rate from 0.57 to 0.81 and the number of seeds scoring above 80% from 5 to 11 of 20. SGD-layer's raw gradient developed the same skew as Adam's.

3.3 MULTI-AGENT

Cooperative pair. As in Section 3.1, a pair of quadrupeds—trained with a shared policy and objective—was evaluated on a fixed set of 500 episodes drawn from Ψ with two robots (Fig. 13). On average (5 seeds), the pair scored in 94% of episodes by step 300 and 99% by step 600, and reached the ball in 1.73 s and the goal in 7.5 s, versus 3.42 s and 9.8 s for a single agent on the same field. The objective was indifferent to how or whether robots interacted, yet cooperation appeared to emerge—in the behavioral trajectories of Figure 9F–J both robots participated and play resembled "passing."

Competitive setting. A defender was trained from scratch against a frozen, pre-trained cooperative pair, and the pair was similarly evaluated on 500 held-out episodes. The trained defender reduced the pair's scoring rate from 0.98 to 0.38, while an untrained defender—stationary on the field—reduced it only to 0.97. The defender tended to transport the ball toward, and often beyond, the field boundary and then park its body between the ball and the pair. This behavior was consistent with the defender's objective to maximize transport loss $\mathcal{T}$, which outweighed the bounds penalty $\mathcal{B}$ tuned for the goal-scoring settings. By contrast, goals could only be scored from within the field, so the pair's objective—minimizing $\mathcal{T}$—aligned with $\mathcal{B}$ and produced delicate play near the boundary, exactly where the defender took the ball. The pair was unable to adapt: fine-tuning for 300 steps against the frozen defender recovered none of its lost performance. To remedy this, we modified the rules of the game to impose a hard boundary—a ball that crossed the boundary was placed back in the field near its exit at the next window boundary (within 1.5 s). Retraining the defender under this rule reduced the pair's scoring rate from 0.99 to 0.77, and with fine-tuning, the pair recovered to 0.89.

4 DISCUSSION

In this work we have demonstrated the use of bare analytic policy gradients for closed-loop, contact-rich loco-manipulation in soft robots (Figs. 1, 9) with thousands of actuated springs (Fig. 6), across multiple morphologies (Fig. 10) and in multi-agent settings (Sect. 3.3; Figs. 9, 13). Training was fast, with single-agent policies requiring only 100 gradient steps, under a minute on a commodity GPU (Sect. 3.1; Fig. 4). It is surprising that our method—absent a learned critic, blend of zeroth-order estimates, or stochastic smoothing—worked. We observed misaligned contact gradients that exhibited up to two orders of magnitude spread in energy (Sect. 3.2; Figs. 11, 12), yet large batches aggregated into a usable signal (Fig. 5). The primary challenge to plain gradient descent was a skew in update energy across network layers (Fig. 14). Both per-coordinate normalization (Adam) and per-layer rescaling by parameter share (SGD-layer) circumvented it, achieving high performance.

This work was not without limitations. Most notably, simulated robots were not transferred to reality. Soft quadrupeds have been realized with cable-driven (Bern et al., 2019) and pneumatic (Drotman et al., 2021) actuators; however, transferring closed-loop loco-manipulation policies to such platforms remains an open problem. Simulation results were themselves limited to a single, penalty-based contact model, and whether our findings generalize to hard, complementarity-based contact is untested. Similarly, we studied one physical model of robot and ball—mass-spring bodies and a rigid sphere—and a simple controller architecture. Finally, our gradient analysis was descriptive: we did not mechanistically characterize the source of gradient magnitude variation, misalignment, or layer-wise skew. Future work could investigate how varying the system's stiffness alters gradients, and whether the skew was architectural or arose from high-dimensional soft-body dynamics.

Future directions also include brain–body co-design, made tractable here by the speed of policy training: at under a minute per policy, morphology could be searched in an outer evolutionary loop. The differentiable simulator also exposes gradients with respect to body parameters, so morphology and control could be optimized jointly by descent instead. Promising emergent behavior in the multi-agent settings may motivate training larger teams and the application of differentiable-game methods, which optimize agents simultaneously rather than by turn-taking, to embodied, contact-rich robotics.

REFERENCES

Rika Antonova, Jingyun Yang, Krishna Murthy Jatavallabhula, and Jeannette Bohg. Rethinking optimization with differentiable simulation from a global perspective. In *Proceedings of the Conference on Robot Learning (CoRL)*, pp. 276–286, 2023.

James M Bern, Pol Banzet, Roi Poranne, and Stelian Coros. Trajectory optimization for cable-driven soft robot locomotion. In *Robotics: Science and Systems (RSS)*, 2019.

Jagdeep Bhatia, Holly Jackson, Yunsheng Tian, Jie Xu, and Wojciech Matusik. Evolution gym: A large-scale benchmark for evolving soft robots. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 2201–2214, 2021.

Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 167–174, 2013.

Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. Scalable co-optimization of morphology and control in embodied machines. *Journal of The Royal Society Interface*, 15(143): 20170937, 2018.

Matteo Cianchetti, Cecilia Laschi, Arianna Menciassi, and Paolo Dario. Biomedical applications of soft robotics. *Nature Reviews Materials*, 3(6):143–153, 2018.

Dylan Drotman, Saurabh Jadhav, David Sharp, Christian Chan, and Michael T Tolley. Electronics-free pneumatic circuits for controlling soft-legged robots. *Science Robotics*, 6(51):eaay2627, 2021.

Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Ignat Georgiev, Krishnan Srinivasan, Jie Xu, Eric Heiden, and Animesh Garg. Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

Jonathan Hiller and Hod Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2012.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics*, 38(6), 2019a.

Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 6265–6271, 2019b.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B. Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark with differentiable physics. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

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

William M Kier and Kathleen K Smith. Tongues, tentacles and trunks: the biomechanics of movement in muscular-hydrostats. *Zoological Journal of the Linnean Society*, 83(4):307–324, 1985.

Sangbae Kim, Cecilia Laschi, and Barry Trimmer. Soft robotics: a bioinspired evolution in robotics. *Trends in Biotechnology*, 31(5):287–294, 2013.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4): 1853–1859, 2020.

So Kuroki, Tatsuya Matsushima, Jumpei Arima, Hiroki Furuta, Yutaka Matsuo, Shixiang Shane Gu, and Yujin Tang. Collective intelligence for 2D push manipulations with mobile robots. *IEEE Robotics and Automation Letters*, 2023.

Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Transactions On Graphics (TOG)*, 38(4):1–13, 2019.

Pingchuan Ma, Tao Du, John Z Zhang, Kui Wu, Andrew Spielberg, Robert K Katzschmann, and Wojciech Matusik. Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021.

Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020.

Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 7805–7817, 2021.

Rhys Newbury, Jack Collins, Kerry He, Jiahe Pan, Ingmar Posner, David Howard, and Akansel Cosgun. A review of differentiable simulators. *IEEE Access*, 12:97581–97604, 2024.

Rolf Pfeifer, Max Lungarella, and Fumiya Iida. The challenges ahead for bio-inspired 'soft' robotics. *Communications of the ACM*, 55(11):76–87, 2012.

Yiling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Differentiable simulation of soft multibody systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 17123–17135, 2021.

Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521 (7553):467–475, 2015.

Yuki Sato, Changyoung Yuhn, Hiroki Kobayashi, Atsushi Kawamoto, and Tsuyoshi Nomura. Computational co-design of structure and feedback controller for locomoting soft robots. *Structural and Multidisciplinary Optimization*, 68(9):185, 2025.

Pierre Schumacher, Daniel Haeufle, Dieter Büchler, Syn Schmitt, and Georg Martius. DEP-RL: Embodied exploration for reinforcement learning in overactuated and musculoskeletal systems. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Luke Strgar and Sam Kriegman. Accelerated co-design of robots through morphological pretraining. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026.

Luke Strgar, David Matthews, Tyler Hummer, and Sam Kriegman. Evolution and learning in differentiable robots. In *Robotics: Science and Systems (RSS)*, 2024.

Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

Tsun-Hsuan Johnson Wang, Juntian Zheng, Pingchuan Ma, Yilun Du, Byungchul Kim, Andrew Spielberg, Josh Tenenbaum, Chuang Gan, and Daniela Rus. Diffusebot: Breeding soft robots with physics-augmented generative diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 44398–44423, 2023.

Eliot Xing, Vernon Luk, and Jean Oh. Stabilizing reinforcement learning in differentiable multiphysics simulation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An end-to-end differentiable framework for contact-aware robot design. In *Robotics: Science and Systems (RSS)*, 2021a.

Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021b.

Yuang Zhang, Yu Hu, Yunlong Song, Danping Zou, and Weiyao Lin. Learning vision-based agile flight via differentiable physics. *Nature Machine Intelligence*, 7(6):954–966, 2025.

A Appendix

A.1 Additional methods

A.1.1 Figures

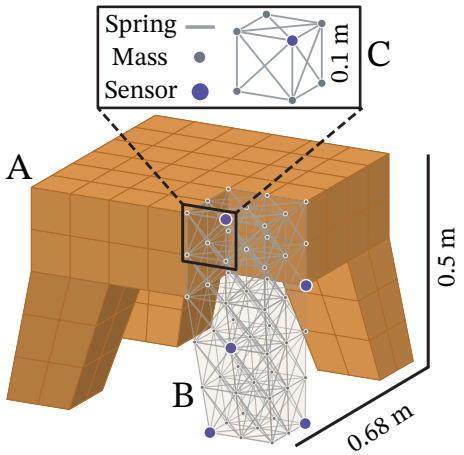

Figure 6: **Soft robot composition.** A quadrupedal soft robot morphology (**A**), shown in its *rest pose*, was modeled as a voxel-based mass-spring lattice (**B**). Each voxel (**C**) comprised eight corner masses linked by edge and face-diagonal springs. Thirty-two sensors, positioned across masses by farthest point sampling, provided egocentric task observations (Sect. 2.3). The quadruped contained 255 masses and 1,538 actuated springs; the biped, 156 and 929; the hexapod, 393 and 2,420.

A.1.2 CONTROLLER DETAILS

Initialization Network weights were initialized from a Kaiming normal distribution (fan-in mode) with gain $g = \sqrt{2}$, except $g = 1$ for the final layer. Layer-normalization weights were initialized to 1 and all layer biases to 0. In the competitive setting, when fine-tuning the pretrained cooperative pair, the input layer was expanded to accommodate observations of the defender. Corresponding weights were initialized to zero so that initial behavior was identical to that of the unmodified network.

Observations The complete set of network observations is provided in Table 1.

Input	Dim	Settings
Sensor distances — Per-sensor 3D range to the listed target. Mean-centered within each target group and normalized by the mean XY distance of the sensors from their centroid.		
ball	32	S, C, K, D
goal	32	S, C, K, D
teammate	32	C, K
opponent(s)	32 (64)	K, (D)
field center	32	S, C, K, D
Ranges — Center-of-mass XY distance between the named pair. Normalized by the field radius, 3.5 m.		
robot–ball	1	S, C, K, D
ball–goal	1	S, C, K, D
robot–field-center	1	S, C, K, D
robot–teammate	1	C, K
teammate–ball	1	C, K
teammate–goal	1	C, K
robot–opponent(s)	1 (2)	K, (D)
opponent–ball(s)	1 (2)	K, (D)
opponent–goal(s)	1 (2)	K, (D)
Velocities — Closing and bearing rates among task objects, and the robot's speed. Normalized by 2 m/s (linear) and 2 rad/s (angular).		
robot–ball closing rate	1	S, C, K, D
robot–ball bearing rate	1	S, C, K, D
ball–goal closing rate	1	S, C, K, D
teammate–ball closing rate	1	C, K
opponent(s)–ball closing rate	1 (2)	K, (D)
robot center-of-mass XY speed	1	S, C, K, D
Clock — Sinusoidal gait reference: $\sin(\omega t + 2\pi k/10)$, $k = 0, \ldots, 9$, $\omega = 15$ rad/s. Teammate robots' (C, K) clocks in antiphase. Unnormalized.		
clock signals	10	S, C, K, D
Total input dimension	S: 113	C: 149 K: 185 D: 185

Table 1: **Network observations.** The "Settings" column indicates presence of each input in the single-agent (S), cooperative multi-agent (C), competitive goal-scoring pair (K), and competitive defender (D) networks.

A.1.3 OPTIMIZER DETAILS

SGD variants Let $g = \nabla_\theta \mathcal{L}$ and η the learning rate. SGD-norm computes the following update:

$$\Delta\theta = -\eta \, \frac{g}{\|g\|_2}.$$

SGD-layer partitions θ into its parameter tensors θ_t—each weight matrix, bias, and layer-normalization gain and shift separately—with d_t parameters each $\left(\sum_t d_t = d\right)$ and computes

$$\Delta\theta_t = -\eta \, \sqrt{\tfrac{d_t}{d}} \, \frac{g_t}{\|g_t\|_2}.$$

Because each per-tensor direction $g_t/\|g_t\|_2$ has unit norm, the concatenated update has norm $\eta\sqrt{\sum_t d_t/d} = \eta\sqrt{1} = \eta$, matching SGD-norm's at the same learning rate (Tab. 2). The two optimizers differ only in how update energy is distributed across the network.

Learning-rate selection Initial learning rates were first tuned by hand, then swept around those values in the single-agent setting. The learning rate with the highest evaluation performance (scoring rate) was selected and reused in the multi-agent settings, except when fine-tuning the cooperative pair's policy, which used an initial learning rate of 0.002.

Optimizer	Values swept	Selected
Adam	0.002, 0.007, 0.02	0.007
SGD	0.07, 0.2, 0.7, 2, 7	0.7
SGD-norm	1.5, 3, 6	3
SGD-layer	1.5, 3, 6	3

Table 2: **Initial learning rates.**

Gradient clipping In the competitive setting networks were trained and fine-tuned with global gradient clipping set at 20. In all other settings, clipping was not applied.

A.2 ADDITIONAL RESULTS

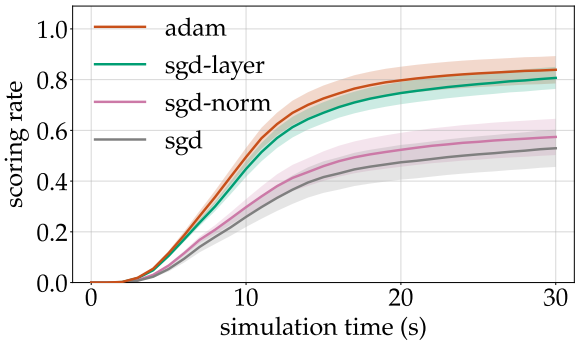

Figure 7: **Scoring rate over episode time.** Cumulative fraction of evaluation episodes ending in a goal (Sect. 3.1; Fig. 4A) as a function of simulation time, for the final policies trained by each optimizer (mean ± standard error over 20 seeds). Few additional goals accrued in the final 10 s of the 30 s cap.

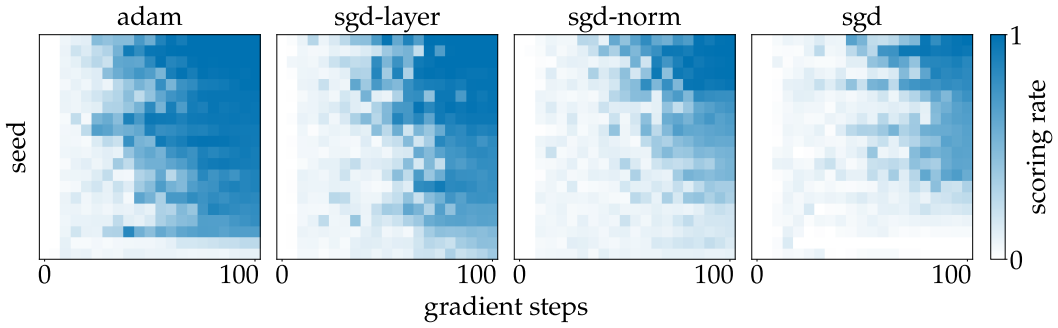

Figure 8: **Per-seed scoring rate.** Scoring rate over gradient steps for each of the 20 seeds in Figure 4, sorted by final value within each optimizer. The gap between optimizers stems from reliability.

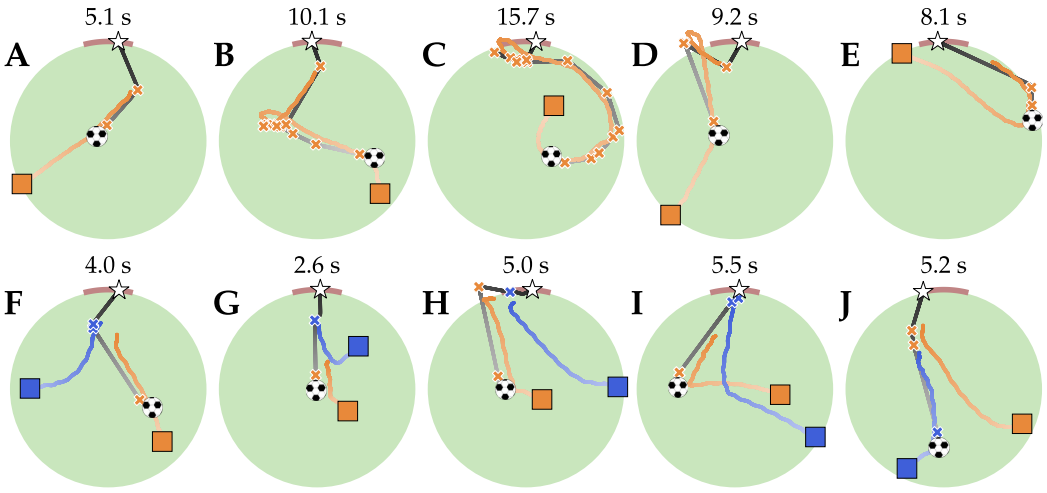

Figure 9: **Behavioral trajectories.** Evaluation episodes played by single-agent (**A**–**E**) and cooperative-pair (**F**–**J**) final policies. Each scene is rotated to place the goal (red arc) at the top. Squares mark robots' initial positions and the checkered icon the ball's. Traces show each robot's and the ball's center of mass, darkening over episode time. Crosses (×) on the ball's trajectory mark robot–ball contacts, and the white star a goal. Time to goal is given above each panel. Behaviors range from a simple approach and kick (A) to multi-touch, cross-field dribbling (B, C), retrieval of missed shots (D), and dexterous handling of the ball near the boundary line (E). In pairs (F–J), the nearer robot struck the ball first and its teammate finished the play, collecting a ball played into its path or a shot that fell short or wide.

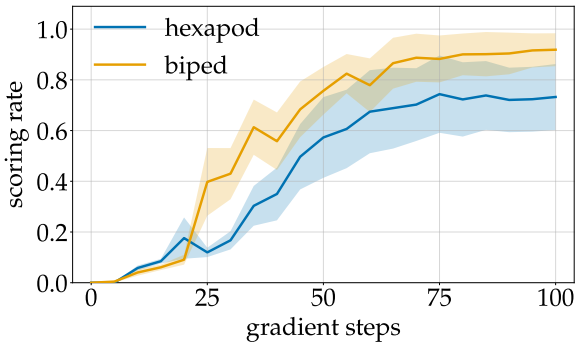

Figure 10: **Extension to other morphologies.** Scoring rate over gradient steps for a hexapod and a biped trained with Adam. Curves show mean ± standard error over 5 seeds; evaluation as described in Section 3.1. Each morphology's evaluation set was drawn independently from Ψ (Sect. 2.5), differing only in the minimum separation constraints, which vary with the size of the morphology.

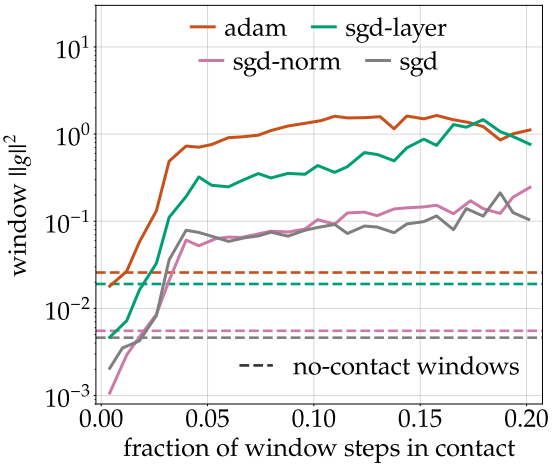

Figure 11: **Gradient energy rises with contact.** Per-window gradient energy $\|g\|^2$ versus the fraction of window steps with robot–ball contact, binned by that fraction (median per bin, 30 bins) and pooled over one run per optimizer. Dashed lines mark the median energy of no-contact windows. Windows in sustained contact ($> 5\%$ of steps) exhibited $\approx$ 10–80$\times$ the energy of no-contact windows across all optimizers.

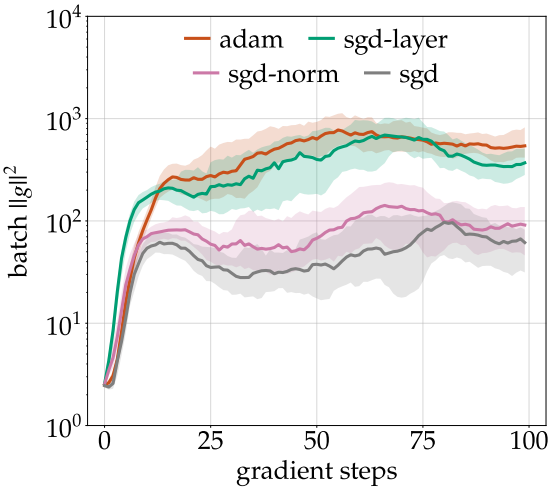

Figure 12: **Batch gradient energy rises early in training.** Energy $\|g\|^2$ of the batch-aggregate gradient at each step (median and interquartile band; 20 seeds) rose by 1.5–2.5 orders of magnitude as locomotion was learned and contact became frequent, then plateaued. Medians and bands smoothed with a 5-step centered moving average.

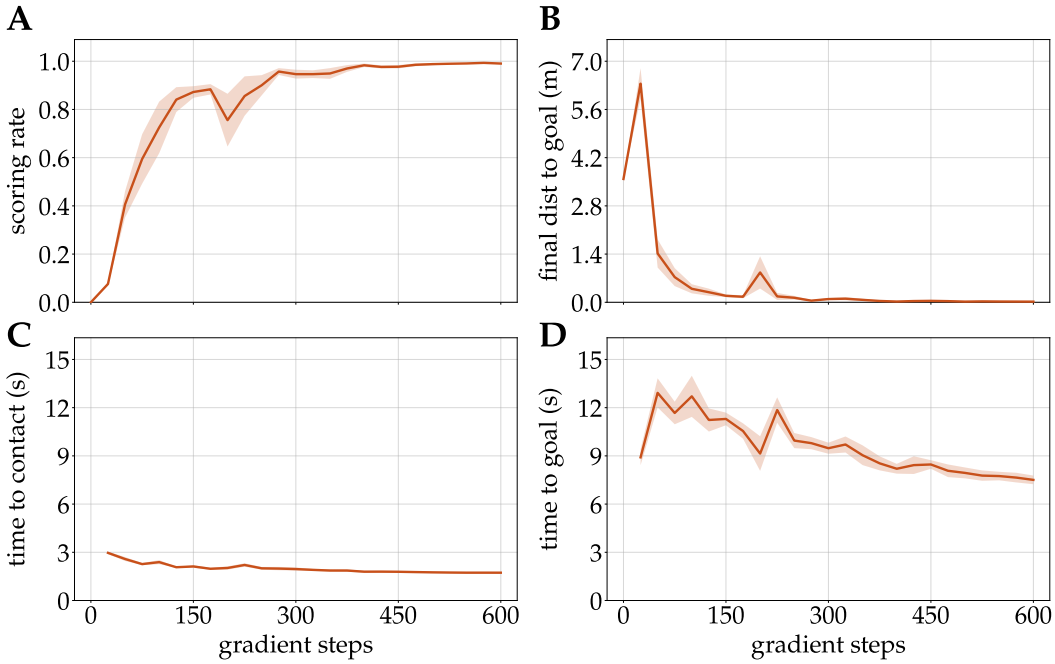

Figure 13: **Performance of a cooperative pair.** Two quadrupeds sharing one policy were trained with Adam for 600 steps. Panels **A–D** as in Figure 4A–D; time to contact is the first touch by either robot. Mean ± standard error over 5 seeds, evaluated as in Section 3.1 on a separate 500-episode set drawn from Ψ with two robots. The second robot helped rather than hindered: pairs scored in nearly every episode by step 300 (A, B), and time to contact (C) and to goal (D) were shorter than in the single-agent setting on the same field, though the pair was also afforded $6\times$ more training.

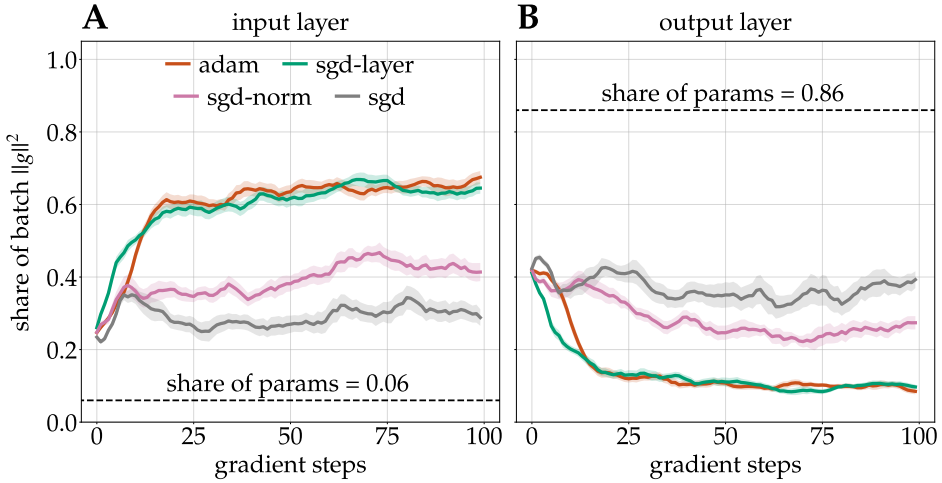

Figure 14: **Gradient energy is skewed across layers.** Fraction of batch gradient energy $\|g\|^2$ held by the controller's input (**A**) and output (**B**) layers over training (mean ± standard error over a 10-seed subset of the runs in Sect. 3.1). Dashed lines mark each layer's share of total network parameters. From initialization, the output layer held an order of magnitude less energy per parameter than the input layer. SGD retained this allocation and SGD-norm shifted it modestly; under Adam and SGD-layer the skew grew much greater. Means and standard error bands smoothed with a 5-step centered moving average.

GENERATIVE AI USAGE

We used generative AI tools to assist in the following: designing and providing feedback on methodology and experiments, implementing methods, refining hypotheses, interpreting results, supporting qualitative and thematic data analysis, identifying relevant literature, drafting and editing this paper for readability, and creating and modifying figures. For all other tasks (see the AI Policy for Authors), generative AI was not used. We reviewed all AI-assisted contributions for correctness and take responsibility for the final contents of this paper.