arXiv:2606.00702v1 [cs.RO] 30 May 2026

Shape Your Body: Value Gradients for Multi-Embodiment Robot Design

Nico Bohlinger[1], **Jan Peters**[1,2]
[1] Technical University of Darmstadt, Germany
[2] Robotics Institute Germany (RIG); German Research Center for AI (DFKI); hessian.AI

nico-bohlinger.github.io/shape-your-body

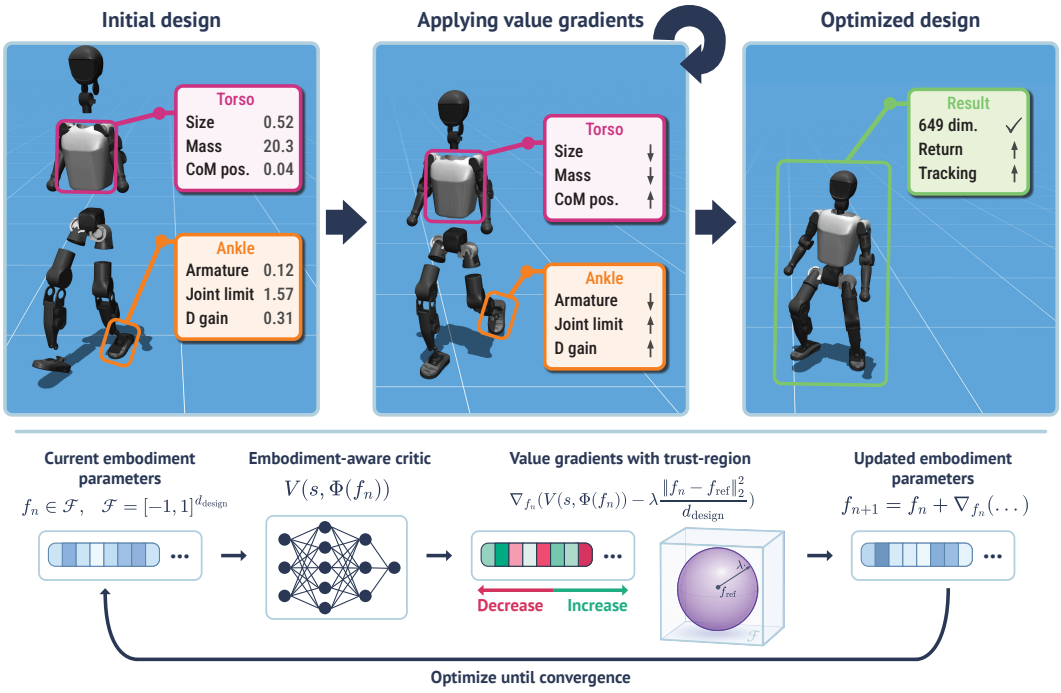


Figure 1: **Shape Your Body.** We first train an embodiment-aware policy and value function with multi-embodiment reinforcement learning, then we optimize new designs by differentiating through the value function and applying the gradients inside a soft trust region around a reference design.

Abstract: We propose to turn generalist multi-embodiment value functions into reusable models for robot design. Instead of running a new reinforcement learning co-design loop for each robot, we first train an embodiment-aware policy and value function across many robot designs. After training, the frozen value function is used as a differentiable surrogate to optimize candidate embodiments through value gradients. We evaluate our approach across different robot design settings, from perturbed single robots to held-out robots across morphology classes, with single models trained on up to 50 robots and design spaces of over 1100 continuous embodiment parameters. Beyond optimizing complete embodiments, we show that value gradients can identify performance-limiting design and control parameters, enabling both the optimization and the analysis of new robot designs.

Keywords: Co-Design, Multi-Embodiment Learning, Reinforcement Learning

1 Introduction

Deep learning methods have revolutionized control and decision-making in robotics by enabling policies to learn complex behaviors in a data-driven manner. In contrast, robot hardware is still largely designed by hand, relying on human engineering expertise. With learning-based control and reliable simulation becoming important design considerations, this manual design process now depends on the intuition of both mechanical engineers and robot learning researchers. As a robot's embodiment fully defines its physical capabilities, even an optimal policy can only be as good as the embodiment allows it to be. Robot design and control are therefore naturally coupled, and jointly optimizing both is commonly referred to as robot co-design [1, 2].

Most co-design methods treat this coupling as a bi-level optimization problem. An outer loop proposes embodiment designs, while an inner loop trains or adapts a controller for each candidate. Evaluating each design often requires full Reinforcement Learning (RL) runs or well-tuned optimal control. Recent RL-based co-design methods reduce this cost by learning embodiment-aware policies and updating design and control within a single training process [3, 4]. However, for every new robot, design space, or task setup, the complete co-design loop must be run again. As a result, scaling co-design to many robots or many design variants remains difficult. For real-world or realistically simulated robots, the design space itself is often high-dimensional, with actuator properties, body-part masses, inertias, and geometry easily reaching hundreds of continuous design parameters.

We instead build on recent progress in learning generalist robot policies across many embodiments [5, 6]. Rather than treating every co-design problem as a new training run, we amortize the process by first training a single multi-embodiment RL policy and value function over a wide distribution of robot designs. The policy learns to control many embodiments, while the value function learns to evaluate how well the shared policy is expected to perform for a given state and embodiment. Once trained, the policy and value function are frozen and reused for downstream design. Because the value function has been trained across many embodiments and morphology classes, it can draw on information from the broader robot distribution when evaluating a candidate design, rather than relying only on data collected for the robot currently being optimized. In addition, downstream design search is computationally cheap, as many candidate robots or design variations can be optimized in parallel through batched value function evaluations, without new expensive RL training runs.

Our experiments study this idea at increasing levels of generalization. We first test our gradient-based design method in the simplest setting, where a value function trained on randomized versions of a single robot is used to recover better designs from perturbed variations. In this setting, we compare against a broad set of derivative-free optimizers that all use the same frozen value function as their surrogate objective. We then move to the more relevant setting where the target robot was not part of the training set, by training a policy and value function on many robots of a morphology class and using it to improve a held-out robot. Finally, we train a single policy and value function across all 50 robots in our dataset, spanning quadrupeds, humanoids, and hexapods, and apply it to robot variations across the morphology classes. This lets us study robot design in substantially higher dimensional spaces than typical robot co-design work, reaching over 1100 continuous embodiment parameters on realistic legged robot models. Besides optimizing complete embodiments, we show that the same value function can be used for design analysis by using its gradients to identify performance-limiting embodiment parameters, such as actuator torque and velocity limits, or body-part masses. These gradients can also provide insight into the control setup, for example by indicating how PD gains should be adjusted for a given robot. Together, these results show that generalist multi-embodiment value functions can act as reusable design models, which we see as a first step toward interactive engineering tools for optimizing and understanding new robot designs.

2 Related Work

In biological systems, morphology and behavior co-develop, i.e., morphology determines which behaviors are possible, while behavioral pressures drive morphological change through evolution [7].

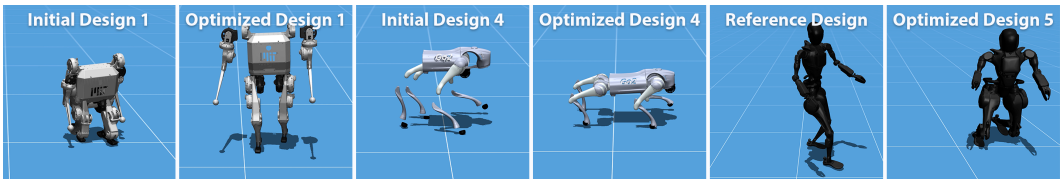


Figure 2: **Optimized robot designs.** We visualize two initial designs and the corresponding optimized designs for the Go2 and the MIT Humanoid. We also show the reference design and an optimized design for the Fourier GR1 T2 humanoid. Although some assets appear stretched or visually disconnected, the underlying geometries remain connected and controllable by the policy. Full optimization trajectories across initial designs for our main three robots are shown in Appendix I.

Robotics inherited this perspective, and through the lens of evolutionary algorithms, foundational work established the embodiment as an optimization variable instead of a fixed constraint when optimizing control [1, 2, 8]. Since then, many works have focused on discrete, graph-based, or grammar-based representations to search over robot structures [4, 9–15], while black-box optimization methods have been used to optimize continuous embodiment parameters or latent embodiment representations [16–19]. These approaches established embodiment optimization as a core robotics problem, but evaluating each design typically requires either costly RL training or potentially suboptimal classical control methods.

A more recent line of work formulates co-design directly as a RL problem, enabling experience sharing across design candidates within a single training process [3, 4, 14, 20–27]. Closest to our use of a value function for design search, Fast Evolutionary Actor-Critic Reinforcement Learning (FEACRL) [3] uses the state-action value function learned by Soft Actor-Critic (SAC) [28] to continuously score candidate designs, which are selected with Particle Swarm Optimization (PSO) [29]. Starting with Transform2Act [4], most recent RL-based methods split episodes into a design phase and a control phase, with dedicated subpolicies. BodyGen [26] improves the phase-specific credit assignment and adds topology-aware representations, while Stackelberg Proximal Policy Optimization (PPO) [27, 30] models the leader-follower dynamics of the two subpolicies. These methods improve the efficiency of the co-design process, but their optimization remains tied to the individual robot and its co-design run. In contrast, we first train the policy and value function over a broad robot distribution, allowing downstream search to use cross-embodiment information while reusing the value function to design many robots or design variants without retraining.

In embodiment-aware learning, the policy and value function can reason about the robot's physical embodiment rather than treating it as a hidden variable. Multi-embodiment learning is a common approach, where a single policy and value function are explicitly or implicitly conditioned on embodiment information. Early methods used Graph Neural Networks (GNNs) to model the robot's kinematic tree [31, 32], while more recent transformer-based architectures treat joints, actuators, or body parts as tokens to improve scalability [16, 33–35]. Other approaches infer the embodiment from proprioceptive history [36, 37], and on-robot learning methods directly train on the real embodiment [38–41]. Recent large-scale multi-embodiment approaches have shown that generalist policies can be trained across broad robot distributions with architectures such as the Unified Robot Morphology Architecture (URMA) [5, 6, 37, 42]. We build on this line of work by using URMA to train a generalist embodiment-aware policy and value function. While prior multi-embodiment work mainly focuses on using the trained policy for control, we use the embodiment-aware value function after training as a reusable surrogate for design optimization and analysis.

Our design search is related to commonly used optimization methods for robot co-design. Classical black-box optimizers such as Bayesian Optimization (BO) [43], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [44], PSO [29], Cross-Entropy Method (CEM) [45], Differential Evolution (DE) [46], Augmented Random Search (ARS) [47], and Trust Region Bayesian Optimization (TuRBO) [48] are commonly used when gradients of the objective are unavailable or unreliable. Recent gradient-based population methods such as Gradient-Covariance Particle Filter Optimiza-

tion (GC-PFO) [19] use the gradient covariance to guide the search in high-dimensional robot design spaces. Our design search operates directly on the trained embodiment-aware value function and therefore has access to the analytic gradients with respect to the high-dimensional embodiment parameters. This is also closely related to explicit policy-conditioned value functions, which learn universal critics conditioned on policy representations, like parameters or embeddings, and update policies by differentiating through the critic with respect to these representations [49–52].

3 Value Gradients for Multi-Embodiment Robot Design

We propose to turn the embodiment-aware value function, obtained from large-scale multi-embodiment RL training, into a reusable surrogate for robot design. The core idea is shown in Figure 1. Instead of running a new co-design RL loop for every target robot, we first train a single embodiment-aware policy and critic across many robot designs. After training, the policy and critic are frozen. The critic is then differentiated with respect to robot design parameters and reused as a design model, so many target embodiments can be optimized with the same frozen networks.

We study robot design for robots with a given kinematic structure, where continuous physical parameters, such as masses, inertias, geometry, joint limits, and actuator properties, can be modified. Each robot is described by a continuous embodiment vector $e \in \mathbb{R}^{d_{\text{phys}}}$ that determines the transition dynamics $p_e(s_{t+1} \mid s_t, a_t)$ and reward function $r_e(s_t, a_t)$. For an embodiment-aware policy $\pi_\theta(a_t \mid s_t, e)$ with parameters θ, the expected discounted return on embodiment e is

$$J(\theta, e) = \mathbb{E}_{\tau \sim \pi_\theta(\cdot|\cdot,e)} \left[\sum_{t=0}^{T} \gamma^t r_e(s_t, a_t) \right], \tag{1}$$

where s_t and a_t are the state and action at time t, $\tau = (s_0, a_0, \ldots, s_T)$ is the trajectory generated by rolling out π_θ under the dynamics p_e, T is the episode horizon, and γ is the discount factor.

We represent each candidate robot design in a normalized design space $\mathcal{F} = [-1, 1]^{d_{\text{design}}}$ and let $f \in \mathcal{F}$ be the normalized design vector. The reference design $f_{\text{ref}} \in \mathcal{F}$ serves as the anchor for the trust region and can be any valid design, such as a nominal URDF for an existing robot or a completely new one. A differentiable map $e = \Phi(f)$ converts the design from the normalized space to physical embodiment parameters such as joint origins and ranges, actuator force and velocity limits, damping, stiffness, armature, friction, PD gains, body masses, inertias, center-of-mass offsets, link geometry, or foot parameters. Hard physical bounds inside Φ can be used to rule out physically implausible or unmanufacturable designs, such as negative masses or overlapping body parts. Design-specific reward terms can also be added to the reward function to discourage plausible but undesirable design features [20]; however, we omit this in our setting. The goal of the design process is to improve an initial design $f_{\text{init}} \in \mathcal{F}$ by iterating on it and finding a design $f^\star \in \mathcal{F}$ with higher return $J(\theta, \Phi(f^\star))$ when rolled out with the frozen policy π_θ.

3.1 Training a Multi-Embodiment Policy and Critic

During RL training, each parallel environment i samples a new design $f_i \sim \mathcal{U}([-c_t, c_t]^{d_{\text{design}}})$ at an episode reset, maps it to a physical robot $e_i = \Phi(f_i)$, and keeps that design fixed for the episode. Designs can come from multiple base robots of different morphology classes. The performance-based curriculum coefficient c_t expands the design support during training and reaches the full design space at $c_t = 1$ [42]. The policy is trained with PPO to maximize

$$\max_\theta \; \mathbb{E}_{f \sim \mathcal{U}([-1,1]^{d_{\text{design}}})} \left[J(\theta, \Phi(f)) \right]. \tag{2}$$

We use URMA as our policy and critic architecture to handle the varying topologies, kinematics, and dynamics across our multi-embodiment training set. For a robot with a set of joints $\mathcal{J}$, URMA splits observations into fixed-size general observations o_g and variable-size per-joint observations $\{o_j\}_{j \in \mathcal{J}}$. Each joint is paired with a description vector d_j that encodes static joint and adjacent body properties, such as joint axis, relative position, limits, gains, body mass, inertia, and geometry.

The same idea is used for feet observations, which are present only in the critic network. URMA encodes each joint separately and aggregates the variable number of latents with an attention scheme:

$$\bar{z}_{\text{joints}} = \sum_{j \in \mathcal{J}} \alpha_j(d_j) \odot g_\psi(o_j), \qquad \alpha_j(d_j) = \frac{\exp(g_\phi(d_j)/\tau)}{\sum_{L_d} \exp(g_\phi(d_j)/\tau)}, \tag{3}$$

where g_ψ and g_ϕ are learned encoders, τ is a learnable temperature, L_d is the latent dimension, and $\odot$ denotes element-wise multiplication. The aggregated joint latent $\bar{z}_{\text{joints}}$, foot latent $\bar{z}_{\text{feet}}$, and general observations o_g are then processed by a shared core network. For the policy, the resulting core latent is decoded back into one action distribution per joint. For the critic, the core latent is decoded into the scalar value prediction.

We modify the standard URMA critic into a direct-design URMA critic to make its value predictions more sensitive to the design parameters. In the original critic, the description vectors d_j only serve as input to the attention keys, while the attention values are computed from the per-joint observations o_j. We extend the encoder g_ψ to take both the observations and the descriptions as input, $g_\psi(o_j, d_j)$, so the embodiment parameters have a more direct influence on the attention values and therefore on the final discounted return prediction, which leads to empirically stronger design gradients. To reduce the sensitivity to approximation errors in the value function, which could lead to catastrophic design updates, we use an ensemble of K core-value heads on top of shared per-joint and per-foot encoders, and optimize their mean prediction. An architecture overview and the ablations that led to the direct-design critic are summarized in Appendix B.

3.2 Value-Gradient Design Search

After multi-embodiment training, we freeze the policy and critic and run Value-Gradient Design Search (VGDS) to optimize the normalized design vector f using gradients of the learned value function. Let $V_k(s, \Phi(f))$ denote the value prediction of critic head k for state s and design f. We combine the K heads into the mean value prediction

$$\bar{V}(s, \Phi(f)) = \frac{1}{K} \sum_{k=1}^{K} V_k(s, \Phi(f)). \tag{4}$$

To make design search cheap and deterministic, we first collect a diverse state bank $\mathcal{S} = \{s_1, \ldots, s_M\}$ from rollouts of the final policy on the full design space $\mathcal{F}$. The critic is trained on designs visited during the multi-embodiment training phase, so its predictions can be unreliable when the search moves too far from the experienced distribution. In preliminary experiments, unconstrained critic maximization often moved designs toward the boundary of the design space, where the critic predicted high returns but the real performance collapsed. We therefore use a soft trust region around a provided reference design f_{ref} to form our main design optimization objective:

$$\hat{J}_\lambda(f) = \frac{1}{M} \sum_{m=1}^{M} \bar{V}(s_m, \Phi(f)) - \lambda \frac{\|f - f_{\text{ref}}\|_2^2}{d_{\text{design}}}. \tag{5}$$

The quadratic penalty discourages designs that move far away from the reference robot, where the critic is more likely to extrapolate. The normalization by d_{design} ensures that the same λ has a comparable effect across robots with differently sized design spaces. The optimizer can still move far from the reference when the gain in predicted value is large enough to compensate for the penalty. We use $\hat{J}_\lambda(f)$ as the objective for Value-Gradient Design Search (VGDS) and for all other search baselines.

VGDS optimizes Equation 5 by gradient ascent on minibatches of the state bank. For a minibatch $B \subset \mathcal{S}$, we optimize

$$\hat{J}_{\lambda,B}(f_n) = \frac{1}{|B|} \sum_{s \in B} \bar{V}(s, \Phi(f_n)) - \lambda \frac{\|f_n - f_{\text{ref}}\|_2^2}{d_{\text{design}}}. \tag{6}$$

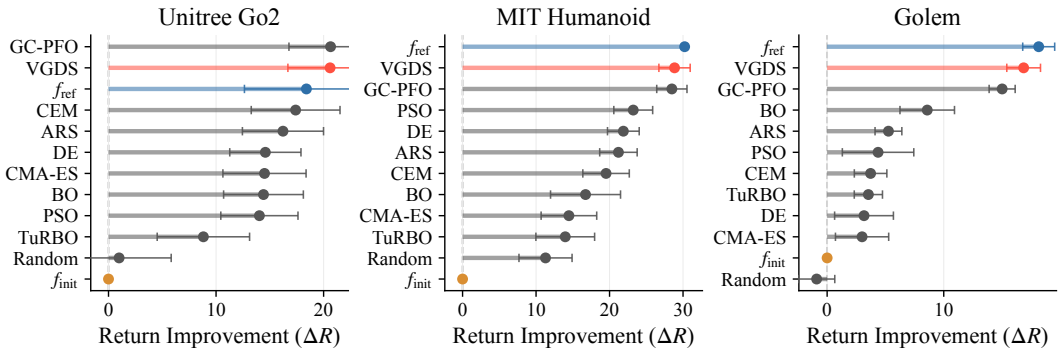


Figure 3: **Single-robot design.** Mean return improvement ΔR over the initial perturbed design f_{init} for 10 starts per robot. The nominal URDF design f_{ref} is shown as a reference, but is only used as the anchor for the trust region.

This leads to the design gradient $g_n = \nabla_f \hat{J}_{\lambda,B}(f_n)$ at iteration n, which contains a value gradient term obtained by backpropagation through the frozen critic and the design map Φ, and a second term through the analytic trust-region gradient $-\frac{2\lambda}{d_{\text{design}}}(f_n - f_{\text{ref}})$. We then apply an Adam [53] ascent step, clip the update of each design parameter to at most $\delta_{\max}$, and finally clip the updated design to the valid design space $\mathcal{F}$:

$$f_{n+1} = \text{clip}_{[-1,1]}\left(f_n + \text{clip}_{[-\delta_{\max},\delta_{\max}]}\left[\Delta_{\text{Adam}}(g_n)\right]\right). \tag{7}$$

VGDS finds the final design $f^\star = f_N$ after N iterations of applying Equation 7. Finally, additional variants of VGDS that we tested during the development are summarized in Appendix J.

4 Experiments

To evaluate our method on a high-dimensional design setting with realistic robot models, we use a standard sim-to-real-transferable velocity-tracking locomotion task from previous URMA-based multi-embodiment learning work [5, 6, 42], implemented in RL-X [54] with MuJoCo XLA (MJX) [55]. Across experiments, the policy and critic are trained on distributions drawn from up to 50 base robots spanning 15 quadrupeds, 31 bipeds and humanoids, and 4 hexapods. Further details on the environment, the set of robots, and the RL training setup are given in Appendix A.

We report and compare the different methods using the return improvement $\Delta R = R(f^\star) - R(f_{\text{init}})$, where f_{init} is the design at which search starts and $f^\star$ is the final design found by the optimizer. To also compare the results on a more robotics-relevant metric of the task, we report the reduction in tracking error of the x-y linear velocity command in Appendix E.

4.1 Single-Robot Design

We first evaluate our design algorithm without cross-robot generalization. We train three separate embodiment-aware policies and critics for uniformly sampled random designs from the full design space $\mathcal{F}$ for the Unitree Go2 quadruped with 358 embodiment parameters, the MIT Humanoid with 514 embodiment parameters, and the Golem hexapod with 688 embodiment parameters. We then start to design 10 uniformly sampled designs from $\mathcal{F}$ for each robot and optimize their embodiment parameters. As a first investigation, we ablate the effect of the trust-region penalty coefficient λ in Appendix D, and choose $\lambda = 100$ for all subsequent experiments.

Figure 3 shows that VGDS, like the other search methods, can significantly improve the sampled designs f_{init} across all three robots, and comes close to, or even exceeds, the performance of the robot's nominal design f_{ref}. f_{ref} most often acts as a strong performance reference, as it is an already well-engineered design and all randomizations are sampled from the design space centered

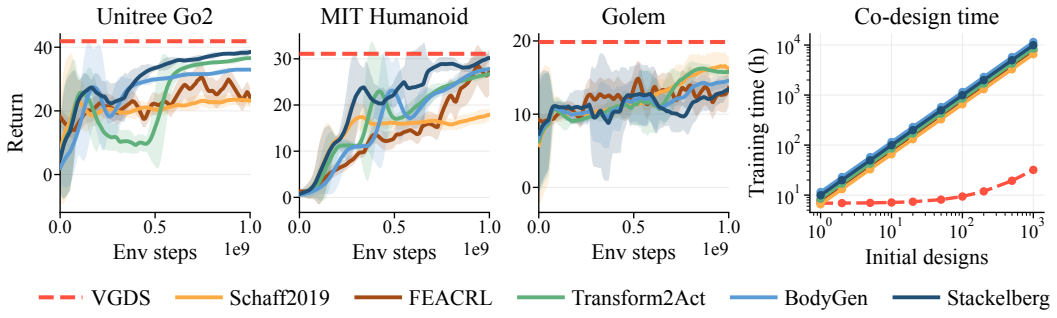


Figure 4: **Comparison to RL-based co-design.** The first three plots show the return over environment steps for Schaff2019 [21], FEACRL, Transform2Act, BodyGen, and Stackelberg PPO. The dashed line shows the final performance of VGDS after training a policy and critic on the full design space, and then designing from the same f_{init} as the RL baselines started from. The fourth plot shows the cumulative time needed to co-design increasing numbers of initial designs.

around it. On the Unitree Go2 and MIT Humanoid, VGDS matches the strongest baseline, and on Golem, it gives the largest improvement of all methods. Across all experiments, GC-PFO proves to be the strongest baseline, as it is the only other method that leverages gradient information, which shows the effectiveness of using the value gradients in such high-dimensional design spaces.

We also compare VGDS to RL-based co-design methods that train a new policy and find a new design for each f_{init}. We substantially adapt the baselines to our setting using URMA-style policies and critics to handle the different morphologies and high-dimensional embodiment parameters, and train them all with PPO. Further details about the adaptations of the baselines to our setting are given in Appendix F. Figure 4 shows that VGDS reaches performance on par with or slightly better than the adapted RL baselines. This difference in training efficiency becomes important when many initial designs have to be optimized. The adapted RL baselines scale linearly with the number of designs, as every initial design requires a separate training run, whereas our approach reuses the same policy and critic. After the initial training run of about 7–9 hours, designing each additional design takes only about 1–2 minutes of search time. This time can be even further reduced by simply batching the critic inference of multiple designs together.

4.2 Generalization Across Training Distributions

Next, we evaluate cross-embodiment generalization of VGDS by training the policy and critic on multiple base robots with the usual embodiment randomization on top. We create training sets for each morphology class but remove the target robot from the training set. Finally, we create one training set that includes all 50 robots, including the target robot. Both settings use the same uniformly sampled random initial designs f_{init} for the design process, which are sampled from the full design space $\mathcal{F}$ of the target robot, as in the previous experiment.

Figure 5 shows that VGDS remains the strongest search method. When trained on all 50 robots, VGDS can find designs that reach higher performance than f_{ref} for the Go2 and the Humanoid, which is a significant improvement over the single-robot training and shows the benefit of training on a broader distribution of robots. For Golem, the trend is different, as the search methods using the hexapod-only training set reach higher improvement than the ones trained on all robots. This is likely due to the hexapod class consisting of only 4 robots, which means the full training set is dominated by quadrupeds and bipeds, and the critic trained on all robots is less specialized to the hexapods. Additionally, we provide the design results for all 50 robots in Appendix G.

4.3 Design Analysis with Value Gradients

We inspect what VGDS changes by grouping the updates $f^\star - f_{\text{ref}}$ by body part and parameter type. Besides physical parameters such as masses or geometry, we look at how VGDS can also pro-

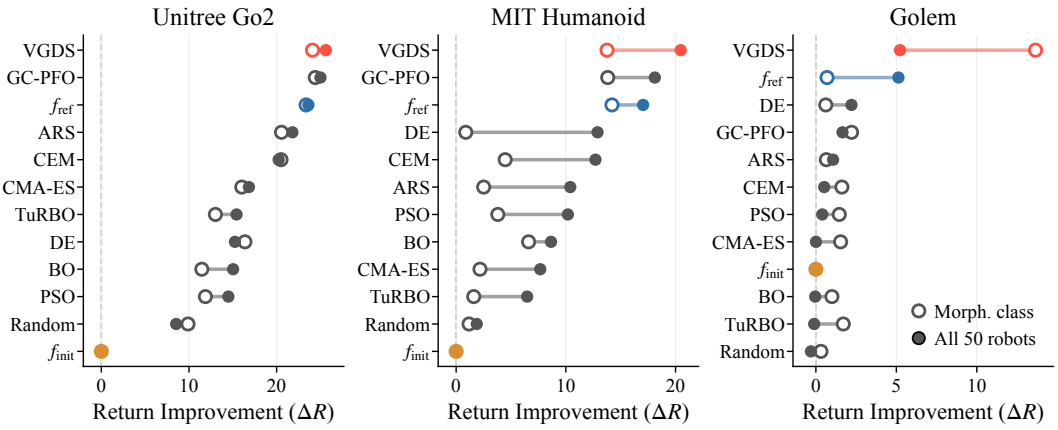


Figure 5: **Effect of training sets.** Target robot is either held out from a morphology class set (open circles) or included in the full 50-robot training set (filled circles). We omit error bars for readability.

vide insights into tuning control parameters. For the MIT Humanoid, the overall strongest changes are nominal joint positions and gains, together with reduced foot size. Copying only the optimized gains into the initial design improves the return already from 5.8 to 12.5 (see Figure 12). For Golem, VGDS significantly reduces the action scale, lowers the P gain, and increases the D gain. For the Unitree Go2, the most prominent changes are more physical and local, including rear leg joint axis changes, foot geometry changes, and different actuator velocity limits on the front hip and calf. Additional evaluations in Appendix H show that no single parameter group explains the full improvement, and the final designs come from coupled updates across many parameters. Lastly, Figure 2 shows an interesting observation for many tall humanoids, such as Fourier GR1 T2, where VGDS makes the robots visibly smaller, wider, and more compact than the reference design, which appears to improve stability and may move them closer to the center of the humanoid training distribution.

5 Limitations

VGDS currently optimizes continuous parameters of a fixed robot topology and does not add or remove joints or body parts, as there is no gradient for topology changes through the URMA network. The method also depends on the coverage and accuracy of the policy and value function. While the trust region helps with errors in extrapolation, having a sufficiently good reference design available is not always possible. Integrating fine-grained design priors and co-design in latent spaces might be ways to mitigate this. Finally, all experiments are performed in rather simple MuJoCo simulations. We therefore do not evaluate whether the optimized designs are manufacturable or transferable to hardware. Checking this properly would require additional constraints or simulators for materials, electronics, actuation, and fabrication, which continues to be an open challenge for robot co-design.

6 Conclusion

We introduced *Shape Your Body*, a method for amortized robot design leveraging gradients from generalist multi-embodiment value functions. After training a policy and value function across many randomized robot embodiments, VGDS reuses the value function as a surrogate design model and optimizes hundreds of continuous embodiment parameters within minutes. Across quadrupeds, humanoids, and hexapods, VGDS significantly improves sampled designs, and matches or exceeds adapted RL co-design baselines while having a much lower marginal cost per additional design. Beyond performance optimization, we see a path toward interactive co-design tools that can optimize a candidate robot and point engineers to relevant changes in gains, actuator limits, geometry, and other parameters. Future work should extend to topology changes, the integration of more complex design constraints in $\Phi(f)$, task-specific design objectives, and trust regions in latent spaces.

Acknowledgments

This project was funded by National Science Centre Poland in the Weave programme UMO-2021/43/I/ST6/02711, and by the German Science Foundation (DFG) under grant number PE 2315/17-1.

References

[1] K. Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994.

[2] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[3] K. S. Luck, H. B. Amor, and R. Calandra. Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning. In *Conference on Robot Learning*, pages 854–869. PMLR, 2020.

[4] Y. Yuan, Y. Song, Z. Luo, W. Sun, and K. Kitani. Transform2act: Learning a transform-and-control policy for efficient agent design. *arXiv preprint arXiv:2110.03659*, 2021.

[5] N. Bohlinger, G. Czechmanowski, M. Krupka, P. Kicki, K. Walas, J. Peters, and D. Tateo. One policy to run them all: an end-to-end learning approach to multi-embodiment locomotion. *Conference on Robot Learning*, 2024.

[6] B. Ai, L. Dai, N. Bohlinger, D. Li, T. Mu, Z. Wu, K. Fay, H. I. Christensen, J. Peters, and H. Su. Towards embodiment scaling laws in robot locomotion. *Conference on Robot Learning (CoRL)*, 2025.

[7] R. C. Bertossa. Morphology and behaviour: functional links in development and evolution. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 366(1574):2056–2068, 2011.

[8] G. S. Hornby and J. B. Pollack. Body-brain co-evolution using l-systems as a generative encoding. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 868–875, 2001.

[9] T. Wang, Y. Zhou, S. Fidler, and J. Ba. Neural graph evolution: Towards efficient automatic robot design. *arXiv preprint arXiv:1906.05370*, 2019.

[10] D. Banarse, Y. Bachrach, S. Liu, G. Lever, N. Heess, C. Fernando, P. Kohli, and T. Graepel. The body is not a given: Joint agent policy learning and morphology evolution. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1134–1142. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[11] A. Zhao, J. Xu, M. Konaković-Luković, J. Hughes, A. Spielberg, D. Rus, and W. Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[12] J. Xu, A. Spielberg, A. Zhao, D. Rus, and W. Matusik. Multi-objective graph heuristic search for terrestrial robot design. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 9863–9869. IEEE, 2021.

[13] D. J. Hejna III, P. Abbeel, and L. Pinto. Task-agnostic morphology evolution. *arXiv preprint arXiv:2102.13100*, 2021.

[14] C. Schaff and M. R. Walter. N-limb: Neural limb optimization for efficient morphological design. *arXiv preprint arXiv:2207.11773*, 2022.

[15] K. Qiu, W. Pałucki, K. Ciebiera, P. Fijałkowski, M. Cygan, and Ł. Kuciński. Robomorph: Evolving robot morphology using large language models. *arXiv preprint arXiv:2407.08626*, 2024.

[16] C. Yu, W. Zhang, H. Lai, Z. Tian, L. Kneip, and J. Wang. Multi-embodiment legged robot control as a sequence modeling problem. *arXiv preprint arXiv:2212.09078*, 2022.

[17] J. Hu, J. Whitman, and H. Choset. Glso: Grammar-guided latent space optimization for sample-efficient robot design automation. In *Conference on Robot Learning*, pages 1321–1331. PMLR, 2023.

[18] K. Ikemura, Y. Dong, and F. T. Pokorny. Latent diffeomorphic co-design of end-effectors for deformable and fragile object manipulation. *arXiv preprint arXiv:2602.17921*, 2026.

[19] A. Vaish and O. Brock. Identifying inductive biases for robot co-design. *arXiv preprint arXiv:2604.11768*, 2026.

[20] D. Ha. Reinforcement learning for improving agent design. *Artificial life*, 25(4):352–365, 2019.

[21] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *2019 international conference on robotics and automation (ICRA)*, pages 9798–9805. IEEE, 2019.

[22] T. Chen, Z. He, and M. Ciocarlie. Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning. In *Conference on Robot Learning*, pages 1158–1173. PMLR, 2021.

[23] Y. Wang, S. Wu, H. Fu, Q. Fu, T. Zhang, Y. Chang, and X. Wang. Curriculum-based co-design of morphology and control of voxel-based soft robots. In *The Eleventh International Conference on Learning Representations*, 2023.

[24] H. Dong, J. Zhang, T. Wang, and C. Zhang. Symmetry-aware robot design with structured subgroups. In *International Conference on Machine Learning*, pages 8334–8355. PMLR, 2023.

[25] M. Li, D. Matthews, and S. Kriegman. Reinforcement learning for freeform robot design. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8799–8806. IEEE, 2024.

[26] H. Lu, Z. Wu, J. Xing, J. Li, R. Li, Z. Li, and Y. Shi. Bodygen: Advancing towards efficient embodiment co-design. In *The Thirteenth International Conference on Learning Representations*, 2025.

[27] Y. Dai, Y. Wang, D. R. Ashley, and J. Schmidhuber. Efficient morphology–control co-design via stackelberg PPO. In *The Fourteenth International Conference on Learning Representations*, 2026.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[29] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

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

[31] T. Wang, R. Liao, J. Ba, and S. Fidler. Nervenet: Learning structured policy with graph neural networks. In *International conference on learning representations*, 2018.

[32] W. Huang, I. Mordatch, and D. Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pages 4455–4464. PMLR, 2020.

[33] A. Gupta, L. Fan, S. Ganguli, and L. Fei-Fei. Metamorph: learning universal controllers with transformers. In *International Conference on Learning Representations*. ICLR, 2022.

[34] A. Patel and S. Song. Get-zero: Graph embodiment transformer for zero-shot embodiment generalization. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14262–14269. IEEE, 2025.

[35] C. Sferrazza, D.-M. Huang, F. Liu, J. Lee, and P. Abbeel. Body transformer: Leveraging robot embodiment for policy learning. In *Conference on Robot Learning*, pages 3407–3424. PMLR, 2025.

[36] M. Liu, D. Pathak, and A. Agarwal. Locoformer: Generalist locomotion via long-context adaptation. In *Conference on Robot Learning*, pages 532–546. PMLR, 2025.

[37] D. Li, B. Ai, N. Bohlinger, J. Peters, H. I. Christensen, and H. Su. Online embodiment adaptation for quadrupedal locomotion. 2026.

[38] L. Smith, I. Kostrikov, and S. Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. In *Robotics: Science and systems*, 2023.

[39] L. Smith, Y. Cao, and S. Levine. Grow your limits: Continuous improvement with real-world rl for robotic locomotion. In *International conference on robotics and automation*, pages 10829–10836. IEEE, 2024.

[40] J. Levy, T. Westenbroek, and D. Fridovich-Keil. Learning to walk from three minutes of real-world data with semi-structured dynamics models. In *Conference on robot learning*, 2024.

[41] N. Bohlinger, J. Kinzel, D. Palenicek, L. Antczak, and J. Peters. Gait in eight: Efficient on-robot learning for omnidirectional quadruped locomotion. *International Conference on Intelligent Robots and Systems*, 2025.

[42] N. Bohlinger and J. Peters. Multi-embodiment locomotion at scale with extreme embodiment randomization. *arXiv preprint arXiv:2509.02815*, 2025.

[43] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

[44] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[45] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*, volume 133. Springer, 2004.

[46] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[47] H. Mania, A. Guy, and B. Recht. Simple random search of static linear policies is competitive for reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[48] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

[49] J. Harb, T. Schaul, D. Precup, and P.-L. Bacon. Policy evaluation networks. *preprint arXiv:2002.11833*, 2020.

[50] F. Faccio, L. Kirsch, and J. Schmidhuber. Parameter-based value functions. In *International Conference on Learning Representations*, 2021.

[51] F. Faccio, A. Ramesh, V. Herrmann, J. Harb, and J. Schmidhuber. General policy evaluation and improvement by learning to identify few but crucial states. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*, 2022.

[52] N. Bohlinger and J. Peters. Massively scaling explicit policy-conditioned value functions. *arXiv preprint arXiv:2502.11949*, 2025.

[53] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[54] N. Bohlinger and K. Dorer. Rl-x: A deep reinforcement learning library (not only) for robocup. In *Robot World Cup*, pages 228–239. Springer, 2023.

[55] 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*, pages 5026–5033. IEEE, 2012. doi:10.1109/IROS.2012.6386109.

[56] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

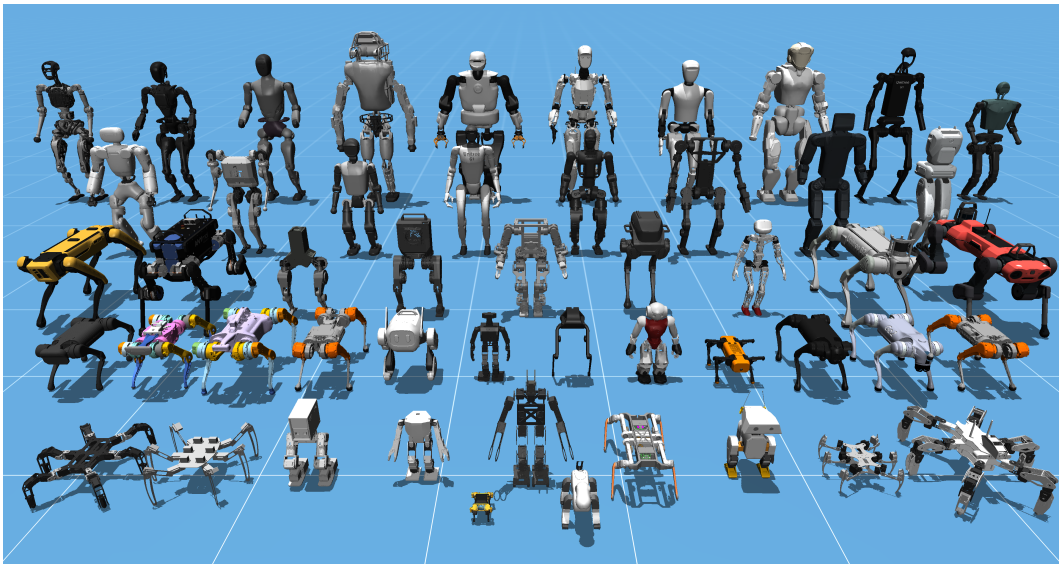


Figure 6: **Overview of all 50 robots used in the multi-embodiment RL training [42].**

Appendix

A Experimental Setup Details

A.1 Environment

All robots are simulated in a fully JAX-jitted MJX locomotion environment implemented in RL-X. The environment runs at a 200 Hz simulation frequency with a 50 Hz control frequency, and each episode lasts at most 20 s, corresponding to 1,000 control steps. Episodes terminate early if the root height drops below 80% of the nominal stance height, which is linearly scaled to 0% for a curriculum coefficient of 1.0. We use the same environment setup for all robots, where the only difference is the nominal joint positions and PD gains, while everything else is shared and automatically adjusted to each robot, e.g. through size-, mass-, or number-of-joints-based scaling of rewards or randomization ranges.

Actions and PD controller. The policy outputs clipped actions $a_j \in [-10, 10]$ for each actuated joint. Actions are converted to target joint positions as

$$q_j^{\text{target}} = q_j^{\text{nom}} + \sigma_{\text{robot}} a_j, \tag{8}$$

where q_j^{nom} is a nominal joint position and σ_{robot} is the robot-specific action scaling factor listed in Table 2. The target positions are tracked by a standard PD controller using the p- and d-gains listed in the same table. The gains and action scaling factor are also part of the searchable design space.

Observations and descriptions. The observations of each actuated joint o_j consist of its position relative to the nominal joint position, velocity, previous action, and an indicator for whether the joint should keep the nominal position, e.g., arms of humanoids or all joints when commanded velocities are zero. The observations of each foot o_f consist of a contact flag, time on ground, and time in air, and are only used by the critic. Static embodiment information is encoded through per-joint and per-foot description vectors d_j and d_f. Each joint description d_j contains the relative joint position, joint axis, attached-body mass, attached-body center of mass, attached-body inertia, number of direct child actuator joints, nominal joint position, torque and velocity limits, damping, armature, stiffness, friction loss, joint range, PD gains, action scaling factor, total robot mass, and

robot dimensions. Each foot description d_f contains the relative foot position, foot-body mass, foot-body center of mass, foot-body inertia, foot geometry size and type, PD gains, action scaling factor, total robot mass, and robot dimensions. The policy additionally receives general observations such as the IMU angular velocity, command velocities, projected gravity, PD gains, action scaling factor, total mass, robot dimensions, and trunk mass, trunk center-of-mass position, and trunk inertia. The critic also observes the IMU linear velocity, terrain height, and the current curriculum coefficient.

A.2 Reward function

The reward consists of x-y-yaw velocity-tracking terms (T_1 and T_2), alive rewards (T_3 and T_4), and regularization penalties (T_5 to T_{23}). Let $c_t \in [0, 1]$ be the curriculum coefficient, and let T_i and w_i denote the reward terms and coefficients in Table 1. The unclipped pre-reward is

$$\tilde{r}_t = w_1 T_1 + w_2 T_2 + c_t \left(w_3 T_3 + \sum_{i=5}^{23} w_i T_i \right), \tag{9}$$

and the final reward is

$$r_t = \max(\tilde{r}_t, 0) + c_t w_4 T_4. \tag{10}$$

All coefficients are multiplied by the control timestep Δt.

Table 1: **Reward terms for locomotion training.** Joint-based and foot-based terms are averaged over the corresponding actuated joints or feet to keep the reward scales consistent across different morphologies.

	Term	Equation	Coef.
T1	x-y velocity tracking	$\exp(-\|v_{xy} - c_{xy}\|_2^2/0.25)$	2.0
T2	yaw velocity tracking	$\exp(-(\omega_z - c_\omega)^2/0.25)$	1.0
T3	Alive clipped	1	0.05
T4	Alive unclipped	1	0.05
T5	Z velocity penalty	$-\|v_z\|^2$	2.0
T6	IMU acceleration penalty	$-\|\Delta v_{\text{imu}}/\Delta t\|_2^2$	10^{-4}
T7	Roll-pitch velocity penalty	$-\|\omega_{xy}\|_2^2$	0.05
T8	Roll-pitch position penalty	$-\|\theta_{xy}\|_2^2$	10.0
T9	Joint nominal deviation penalty	$-\overline{\|q_u - q_u^{\text{nom}}\|^2}$	20.0
T10	Joint position limit penalty	$-\overline{[\ell_u - q_u]_+ + [q_u - u_u]_+}$	40.0
T11	Joint velocity limit penalty	$-\overline{[\|\dot{q}_u\| - 0.9\dot{q}_u^{\max}]_+}$	5.0
T12	Joint velocity penalty	$-\overline{\|\dot{q}_u\|^2}$	4×10^{-4}
T13	Joint acceleration penalty	$-\overline{\|\Delta\dot{q}_u/\Delta t\|^2}$	5×10^{-6}
T14	Joint torque penalty	$-\overline{\|\tau_u\|^2}$	4×10^{-4}
T15	Positive power penalty	$-\overline{[\tau_u\dot{q}_u]_+}$	4×10^{-4}
T16	Action rate penalty	$-\overline{\|a_t - a_{t-1}\|^2}$	3.0
T17	Action smoothness penalty	$-\overline{\|a_t - 2a_{t-1} + a_{t-2}\|^2}$	0.1
T18	Self-collision penalty	$-n_{\text{col}}$	2.0
T19	Base-height penalty	$-\|h - h^{\text{nom}}\|^2$	30.0
T20	Foot air-time penalty	$\overline{\mathbb{1}_{c_f} \min(t_f^{\text{air}} - 0.4 s_{\text{robot}}, 0)}$	3.0
T21	Symmetric air penalty	$-\overline{\mathbb{1}_{\neg c_{f_l}} \mathbb{1}_{\neg c_{f_r}}}$	1.0
T22	Foot slip penalty	$-\overline{\mathbb{1}_{c_f} \|v_{xy}^f\|^2}$	0.1
T23	Foot z-velocity penalty	$-\overline{\|\min(v_z^f, 0)\|^2}$	0.2

A.3 Curriculum and design sampling

At each episode reset, every training robot samples a normalized design vector $f \sim \mathcal{U}([-c_t, c_t]^{d_{\text{design}}})$ and keeps this design fixed for the episode. Training starts with a curriculum

coefficient of $c_t = 0$, where only the nominal design is sampled, and expands toward $c_t = 1$, where the full design space is available. The curriculum follows the performance-based embodiment randomization scheme of Bohlinger et al. [42]. All reported evaluation results always use the full design space.

We additionally randomize initial states and apply observation and actuation perturbations during training. This includes a random action delay of up to one control step with 5% mixed-delay probability, joint and IMU observation noise, random initial yaw, small roll-pitch perturbations, joint position perturbations around the nominal pose, and trunk velocity pushes.

A.4 Robots and design dimensions

Figure 6 shows the 50 robots used in the largest training set, which contains 15 quadrupeds, 31 bipeds and humanoids, and 4 hexapods. For each robot, Table 2 reports the default actuator p-gain and d-gain, action scaling factor σ_{robot}, number of actuated joints n_u, and design dimension d_{design}.

Table 2: Set of robots used for multi-embodiment training and design.

Robot	p-gain	d-gain	σ_{robot}	n_u	d_{design}
Quadrupeds					
Unitree A1	20.0	0.5	0.25	12	358
Unitree Go1	20.0	0.5	0.25	12	358
Unitree Go2	20.0	0.5	0.30	12	358
Unitree B2	80.0	2.0	0.50	12	358
ANYmal B	80.0	2.0	0.50	12	358
ANYmal C	80.0	2.0	0.50	12	358
Barkour v0	20.0	0.5	0.25	12	358
Barkour vB	30.0	0.5	0.25	12	358
Silver Badger	20.0	0.5	0.25	13	385
Honey Badger	20.0	0.5	0.25	12	358
Bittle	25.0	0.5	0.60	8	250
Aibo	20.0	0.5	0.25	17	493
Solo 12	20.0	0.5	0.25	12	358
Spot	80.0	2.0	0.50	12	358
Spot Mini	20.0	0.5	0.25	12	358
Bipeds and humanoids					
Bolt	15.0	0.3	0.20	6	190
Open Duck Mini	6.5	0.5	0.50	16	460
Mini Pi	20.0	0.5	0.30	12	352
QMini	15.0	1.0	0.40	10	298
Tron 1	40.0	1.0	0.50	6	190
Unitree H1	60.0	2.0	0.75	19	541
Unitree G1	45.0	1.0	0.50	23	985
Talos	80.0	2.0	0.75	30	1020
ROBOTIS OP3	21.0	0.5	0.60	20	568
NAO v5	30.0	1.0	0.70	22	1102
ADAM Lite	80.0	2.0	0.75	25	703
AgiBot X1	45.0	1.0	0.70	29	811
Apptronik Apollo	100.0	2.0	0.75	30	890
Atlas	80.0	2.0	0.75	27	783
Berkeley Humanoid	20.0	0.6	0.50	12	352
Berkeley Humanoid Lite	20.0	0.5	0.30	22	661
Booster T1	25.0	0.6	0.50	23	649
ELF2	35.0	0.7	0.50	25	703
EngineAI SA01	55.0	1.5	0.50	12	352
Fourier GR1 T2	80.0	2.0	0.75	34	946

Continued on next page.

Robot	p-gain	d-gain	σ_{robot}	n_u	d_{design}
Fourier GR2 v3	80.0	2.0	0.75	29	811
Fourier N1	70.0	2.0	0.70	23	649
K-Bot v1	40.0	2.0	0.50	20	594
K-Bot v2	50.0	1.0	0.50	20	594
MIT Humanoid	30.0	0.5	0.50	18	514
Northwestern Humanoid	20.0	0.6	0.50	10	298
Poppy Humanoid	20.0	0.5	0.30	25	703
Sigmaban	21.0	0.5	0.50	20	568
STAR1	80.0	2.0	0.75	31	1177
Valkyrie	100.0	2.0	0.75	26	730
Z-Bot	10.0	0.5	0.40	20	594
Hexapods					
Crab	20.0	0.5	0.50	18	526
Custom Hexapod	30.0	0.5	0.60	18	526
Tom Hexapod	25.0	0.5	0.50	18	526
Golem	25.0	0.5	0.50	24	688

A.5 PPO Training

All embodiment-aware policies and critics are trained with PPO. We use the hyperparameters in Table 3. Individual experiments deviate mainly in the number of training robots, and therefore in the number of parallel environments and total environment steps.

Table 3: **PPO hyperparameters.** These are the default training hyperparameters. Large multi-robot runs adjust the total number of environment steps and number of parallel environments.

Hyperparameter	Value
Rollout length	64
Minibatch size	16,384
Epochs	10
Learning rate schedule	linear, $1.5\times10^{-3} \rightarrow 3\times10^{-4}$
Discount factor γ	0.99
GAE λ	0.7
Clip range	0.1
Entropy coefficient	0.005
Max gradient norm	3.0
KL threshold	0.5
Initial action standard deviation	1.0
Policy mean clipping	$[-10, 10]$
Policy std. clipping	$[0.01, 2.0]$
Parallel environments	4,096 or 7,680 or 15,040
Total environment steps	1.0×10^9 or 3.7×10^9

Single-robot models are trained on randomized versions of one robot with 4,096 parallel environments for 1.0×10^9 environment steps. Morphology class models are trained on all robots of one morphology class except the held-out target robot, using 7,680 parallel environments for 3.7×10^9 environment steps. The all-robot model is trained on all 50 robots with 15,040 parallel environments for 3.7×10^9 environment steps.

Training was done on a mix of NVIDIA RTX 3090, A5000, A6000, RTX 6000 Ada, and V100 GPUs. The largest experiment with all 50 robots took around 2.5 days on a single RTX 6000 Ada.

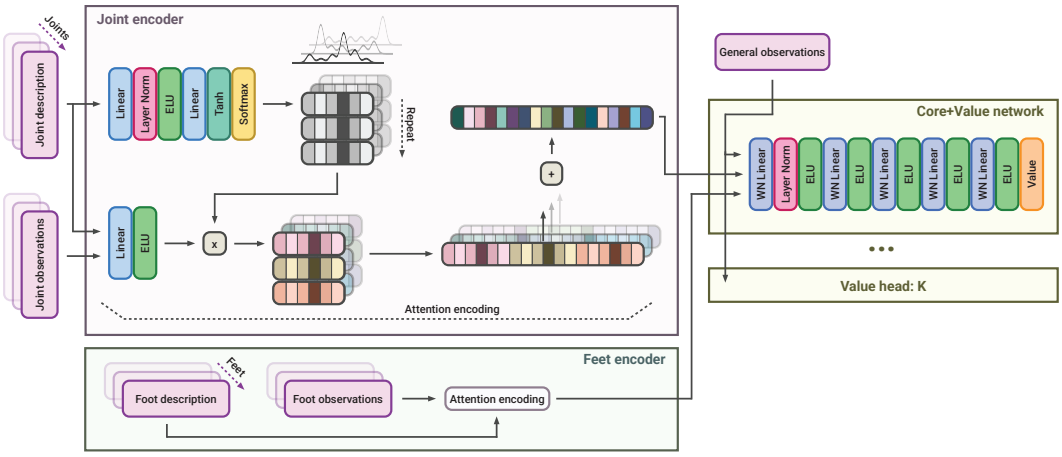


Figure 7: **Overview of the direct-design URMA critic architecture.** We simplify the visualization: The foot encoder corresponds to the joint encoder, but with foot observations and foot description vectors as input, and the value heads all use the same structure. The linear layers in the value heads are all wrapped in WeightNorm (WN) layers [56].

B Direct-Design URMA Critic

Figure 7 shows the direct-design critic used in our experiments. Additionally, Table 5 shows the layer widths and parameter counts. Compared to the original URMA critic, the per-joint and per-foot encoders receive the corresponding description vectors in addition to the observations. This gives the embodiment parameters a direct path into the value prediction.

Table 4: Critic architecture ablation for gradient-based design on an early-stage Unitree Go2 design task, with results averaged over 10 initial designs.

Critic	Heads	Latent dim.	ΔR
Original URMA critic	1	4	+10.39
Original URMA critic	3	4	+13.57
Original URMA critic	3	8	+12.59
Original URMA critic	3	16	+10.22
Direct-design critic	3	4	+6.41
Direct-design critic	3	8	+9.12
Direct-design critic	3	16	+10.98
Direct-design critic	3	32	**+17.76**
Direct-design critic	3	64	+14.35

The ablations in Table 4 show that the direct-design architecture only becomes beneficial once the latent of the attention values has sufficient capacity. With the default URMA latent dimension of 4, the direct-design critic is bottlenecked and underperforms even the original critic. Increasing the latent dimension improves the direct-design critic up to dimension 32, where it gives the strongest design improvement across all variants. Increasing the dimension further does not help, while increasing the original critic also does not close the gap. We therefore use the direct-design URMA critic with $K = 3$ heads and value-latent dimension 32 throughout all our experiments.

C Surrogate Optimizer Hyperparameters

All optimizers use the same frozen direct-design URMA critic and optimize the same design optimization objective $\hat{J}_\lambda(f)$ from Equation 5, with a minibatch size of 64 states from the fixed state bank. We set $\lambda = 100$ for the trust-region penalty. An ablation over λ values is presented in Ap-

Table 5: Layer widths and parameter counts of policy and critic.

Block	Critic	Policy
Joint encoder	64, 64	256, 256
Joint value latent	32	4
Foot encoder	32, 32	—
Foot value latent	32	—
Head MLP	512, 256, 128, 128, 128, 1	512, 256, 256, 256, 256
# heads	3	1
Total parameters	5,378,312	945,430

pendix D. All final results are evaluated by rolling out the frozen policy π_θ on the found design $f^\star$ for 200 episodes and reporting the mean return.

We tuned the optimizer-specific hyperparameters on an early-stage version of the perturbed-start experiment on all three robots and made sure that all optimizers converged and used roughly similar budgets with respect to the number of iterations and critic evaluations. Their hyperparameters are as follows:

Value-Gradient Design Search. VGDS optimizes Equation 5 with default Adam for 50 iterations. The learning rate is 0.05 and each normalized design-parameter update is clipped to $\delta_{\max} = 0.1$.

Random search. Random search evaluates 3200 designs sampled uniformly from the normalized design space and returns the design with the highest value of $\hat{J}_\lambda(f)$.

Bayesian optimization. BO uses the BoTorch SingleTaskGP with a Matérn kernel and qLogNoisyExpectedImprovement as the acquisition function. The optimizer starts from 10 Sobol initial points, with the first point set to the initial design, and then runs for 50 Bayesian optimization iterations. The acquisition function is optimized with 8 restarts, 256 raw samples, and batch size $q = 1$.

Covariance matrix adaptation evolution strategy. CMA-ES is run for 50 generations with initial step size $\sigma_0 = 0.5$ and population size $4 + \lfloor 3 \ln(d_{\text{design}}) \rfloor$. We use the diagonal-covariance parameterization due to the high dimensionality of the design space.

Trust-region Bayesian optimization. TuRBO is implemented as adaptive trust-region random search. It runs for 50 iterations with 64 candidate designs per iteration. The trust-box radius is initialized to 0.1, with minimum radius 0.005 and maximum radius 0.5. The shrink and expansion factors are 0.7 and 1.3, respectively.

Particle swarm optimization. PSO uses the global-best variant with 32 particles for 50 iterations. We use inertia $w = 0.9$, cognitive coefficient $c_1 = 0.5$, and social coefficient $c_2 = 0.3$.

Cross-entropy method. CEM is run for 50 iterations with 32 particles. Each iteration samples from a diagonal Gaussian, selects the top 20% elites, and refits the mean and standard deviation to those elites. The initial standard deviation is 0.5, and the standard-deviation floor is 0.02.

Differential evolution. DE is run for 50 iterations with 32 particles. For each particle, we form a mutant vector from three randomly selected particles using differential weight $F = 0.5$, apply binomial crossover with probability $CR = 0.9$, and keep the trial candidate if it improves the objective.

Augmented random search. ARS uses antithetic finite differences with 16 directions for 50 iterations. We use perturbation standard deviation 0.1, learning rate 0.05, and top-direction fraction 1.0.

Gradient-covariance particle filter optimization. GC-PFO is run for 50 iterations with 2 regions and 32 particles per region. At each iteration, particles are resampled according to their objective values and updated with gradient-covariance-shaped exploration noise. We use step size 0.2, initial temperature $\tau_0 = 1.0$, and $\epsilon = 10^{-8}$.

D Trust-Region Sensitivity

The value gradients are only useful if search stays in a region where the value function is reliable. We therefore perform a sweep over the trust-region coefficient λ on the single-robot design setting, where $\lambda = 0$ effectively means no trust-region. Table 6 shows that without the trust-region penalty, performance drops a lot. On the Unitree Go2, the final design is significantly worse than the initial random design, while the MIT Humanoid and Golem improve only slightly. For $\lambda > 0$, performance improves on all three robots. Unitree Go2 performs best at $\lambda = 100$. MIT Humanoid and Golem reach their highest values at larger penalties, but the differences are small compared to the gap between $\lambda = 0$ and any useful trust region. We use $\lambda = 100$ for the main experiments as it is the best value on Unitree Go2, close to the best values on the other robots, and avoids tuning the penalty separately per robot. For the absolute best performance, tuning λ per robot can be a good option for practical applications.

Table 6: **Trust-region sensitivity.** Mean return improvement ΔR of VGDS on the single-robot design setting as a function of the trust-region penalty coefficient λ over 10 initial designs.

Robot	$\lambda = 0$	$\lambda = 5$	$\lambda = 10$	$\lambda = 20$	$\lambda = 50$	$\lambda = 100$	$\lambda = 150$	$\lambda = 200$	$\lambda = 300$
Unitree Go2	-18.93	14.51	14.30	14.18	13.93	**20.60**	14.18	14.12	13.87
MIT Humanoid	2.21	29.08	29.77	30.39	30.95	28.84	31.15	**31.20**	30.97
Golem	0.50	6.03	9.94	9.73	14.08	16.85	15.63	16.46	**17.27**

E Velocity Tracking Error Reduction Results

As a more intuitive and task-specific metric, we also report the reduction in absolute x-y linear velocity tracking error,

$$\Delta e_{xy} = e_{xy}(f_{\text{init}}) - e_{xy}(f^\star), \tag{11}$$

where positive values indicate lower tracking error after the design search. Improvements in return, as shown in Figure 3 and Figure 5, do not always imply lower x-y tracking error, especially when the optimized design trades tracking performance against other reward terms.

In the single-robot setting, Figure 8 closely follows the return results. VGDS and GC-PFO give almost identical tracking error reductions on all three robots and are the strongest search methods overall. For the training set comparison in Figure 9, the tracking metric shows the same broad trend for the three robots.

F RL Co-Design Baselines

We adapt Schaff2019 [21], FEACRL, Transform2Act, BodyGen, and Stackelberg PPO to the same high-dimensional locomotion co-design setting used in our main experiments. The original methods were proposed for different robot representations, lower dimensional design spaces, or other base RL algorithms. The Transform2Act family of methods was originally proposed to also handle discrete topology optimization. In our setting, we only evaluate the continuous-design variant, because our goal is to optimize fixed-topology robot models around plausible initial designs rather than to evolve new robots from scratch. To make all the RL methods compatible, we replace their normal policy and critic networks with URMA-style networks, and train them all in the on-policy setting with PPO. While these adaptations make the comparison controlled, they do not fully reproduce the original methods, and therefore the results should be interpreted as comparisons of the underlying co-design ideas rather than of the specific implementations.

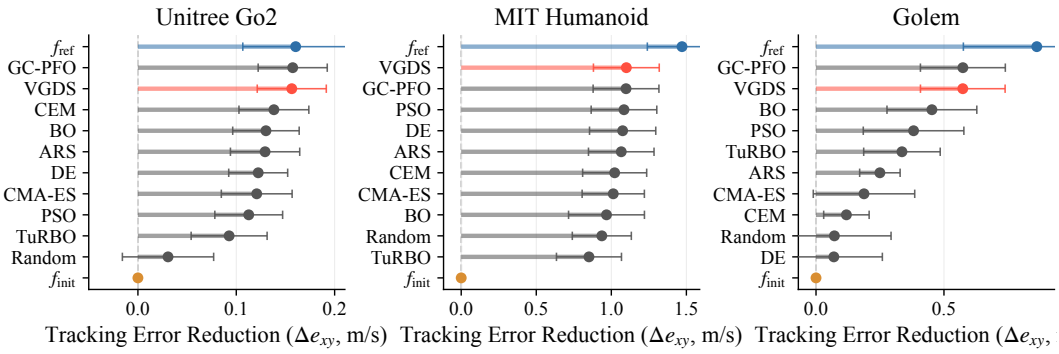


Figure 8: x-y **tracking error reduction for single-robot design.** Mean reduction Δe_{xy} in absolute x-y linear velocity tracking error relative to the initial perturbed design f_{init}, for 10 starts per robot.

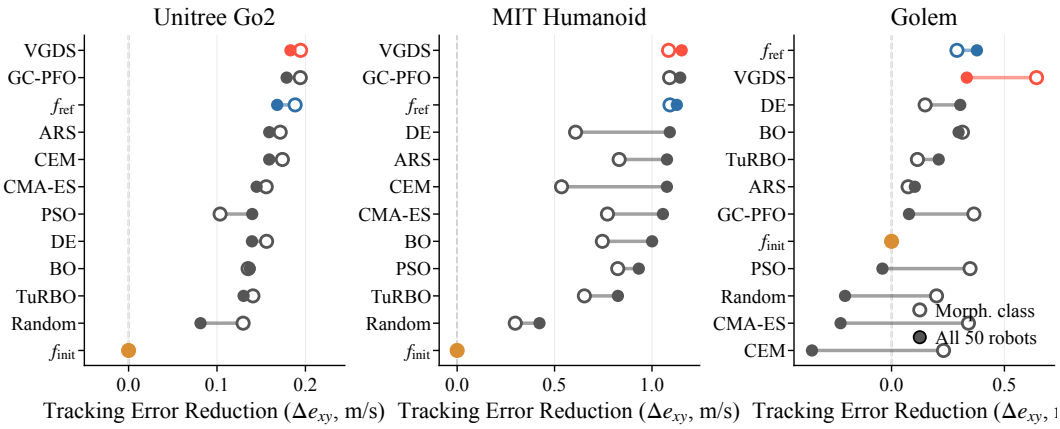


Figure 9: x-y **tracking error reduction for multi-robot design.** Mean reduction Δe_{xy} in absolute x-y linear velocity tracking error when the target robot is either held out from its morphology class training set or included in the full 50-robot training set. We omit error bars for readability.

Schaff2019. Schaff2019 maintains a distribution over designs and updates this distribution from rollout returns while training a design-conditioned policy. We keep this population-based structure, but represent designs in the normalized embodiment space f and train the policy with our PPO implementation. Each parallel environment samples a design from the current design distribution, and the distribution is shifted toward designs that achieve higher episode return.

Fast Evolutionary Actor-Critic Reinforcement Learning. FEACRL alternates between controller learning and design selection, using an actor-critic value estimate to score candidate designs. The original method uses off-policy SAC and a learned state-action value function. In our on-policy implementation, we replace SAC with PPO and use the learned value function over a buffer of start states as the design surrogate. Candidate designs are selected with PSO in the normalized design space.

Transform2Act. Transform2Act learns a transform policy that modifies the robot design before an execution policy controls the resulting body. Our robots have fixed topology and continuous design parameters, so we use the continuous attribute variant and remove the discrete skeleton actions such as adding or removing joints. At the beginning of each episode, the transform policy outputs a continuous design update in the normalized design space and the resulting embodiment is then used

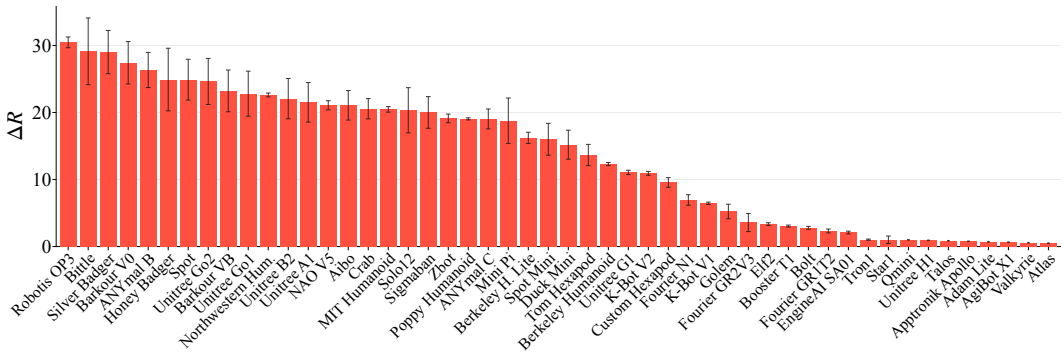


Figure 10: **Design search on all 50 training robots.** Per-robot return improvement $\Delta R = R_{\text{found}} - R_{\text{init}}$ of VGDS applied to each robot in the full 50-robot training set, starting from 10 uniformly sampled random initial designs per robot.

for the rollout of the execution policy. Both the transform and execution policies are trained with PPO.

BodyGen. BodyGen builds on the structure of Transform2Act and improves the architecture and credit assignment between design and control. Because our robot topologies are fixed and the morphology is already represented by URMA description vectors, we do not use BodyGen's additional topology-aware self-attention. However, we implement the stage-specific credit assignment. In our setting, BodyGen therefore differs from Transform2Act mainly through its separate design and control value functions and the corresponding stage-specific advantage estimates.

Stackelberg PPO. Stackelberg PPO builds on BodyGen and treats morphology optimization and control learning as a leader-follower problem, where the design policy acts as the leader and the control policy adapts as the follower. We adapt this idea to the same fixed-topology continuous design space as the other baselines. The leader proposes design updates in the normalized embodiment space, while the follower is a URMA-style PPO policy trained on the resulting designs.

G All 50 Robots Design Results

We use the same trained policy and critic from the main experiments, which were trained on all 50 robots, to run VGDS on every robot in the training set, and report the return improvements in Figure 10. We note that some of the large humanoid runs do not appear to have fully converged yet and would benefit from more training steps, which is the reason for the small improvements on some of those robots.

H Design Analysis Details

We provide additional visualizations for the design analysis in subsection 4.3. We compare optimized designs to the nominal reference in the normalized design space, $\Delta f = f^\star - f_{\text{ref}}$, and group dimensions by body part and parameter type. Each group is summarized by the Root Mean Square (RMS) update magnitude and its dominant signed direction across design searches.

Figure 11 shows the full grouped changes. MIT Humanoid has strong changes in nominal joint positions and foot geometry. Unitree Go2 shows more localized changes in leg axes, foot geometry, and actuator limits. Golem shows large control parameter changes, especially action scale and gains, together with nominal position changes in the coxa.

Figure 12 evaluates selected groups in isolation and in combination, by substituting them into the initial design and evaluating the return improvement. For MIT Humanoid, the optimized gains

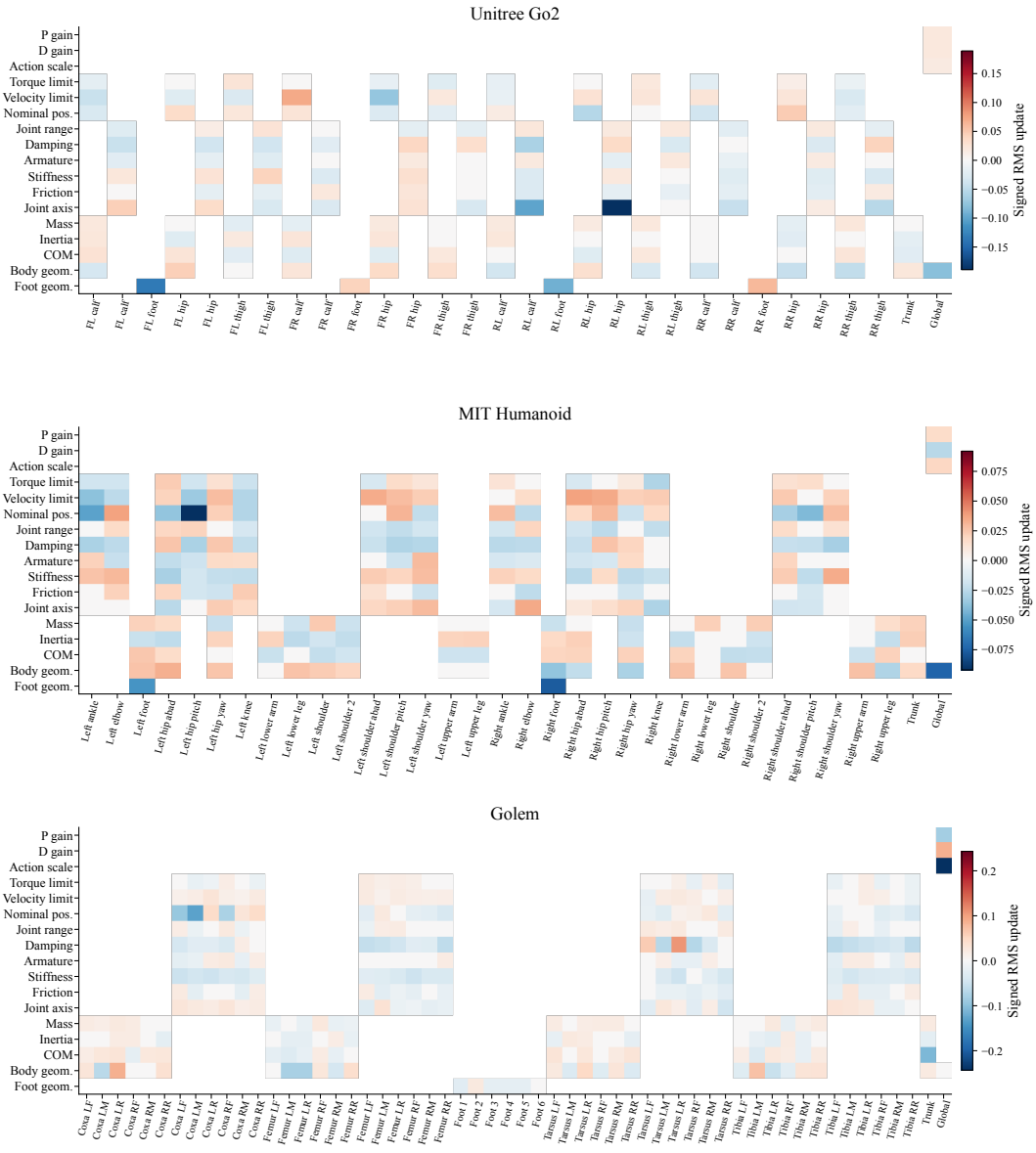


Figure 11: **Grouped design changes.** Signed RMS changes of the optimized designs relative to the nominal reference, grouped by body part and parameter type. The heatmaps show which design groups are changed most consistently by VGDS.

are the most useful isolated group. For Unitree Go2, the best result comes from combining the highlighted joint axis, foot geometry, and actuator velocity limit changes. For Golem, reducing the action scale helps, while changing the PD gains alone hurts performance. In general, the partially modified designs recover part of the full improvement, but the complete optimized design remains substantially better.

As an additional sanity check, we verify that the optimized designs do not simply saturate the design bounds. Across the three single-robot experiments, no optimized dimension exceeds a near-boundary threshold of $|f_i| > 0.9$, and the average normalized change per dimension is small, between 0.027 and 0.042. The largest systematic shift is the Golem's control parameters, where VGDS reduces the action scale by 12.2%, lowers the P gain by 4.1%, and increases the D gain by 4.4%. On the 17 evaluated partially modified designs, the value predictions are positively correlated with true

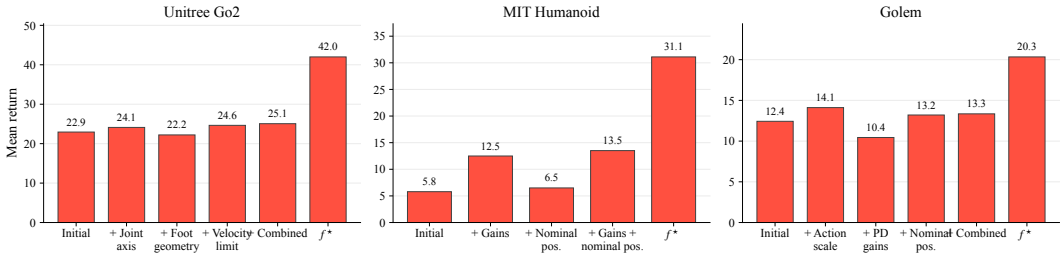


Figure 12: **Parameter group evaluations.** Selected parameter groups from the optimized design $f^\star$ are copied into the initial design f_{init} and evaluated with the same policy.

rollout return, with Spearman $\rho = 0.40$ overall and $\rho = 0.50$, 0.60, and 0.83 for MIT Humanoid, Unitree Go2, and Golem. The final design $f^\star$ has both the highest value prediction and the highest rollout return for all three robots.

I Design Search Trajectories

Figure 13, Figure 14, and Figure 15 show the VGDS optimization trajectories in the single-robot setting for 8 of the 10 initial designs used for the main experiments with the Unitree Go2, MIT Humanoid, and Golem, respectively. While the initial designs all visually appear quite different, VGDS finds consistent improvements across all of them, and the designs all converge to a similar robot-specific optimum. This shows a sort of diversity collapse, which is expected but not always welcome in design optimization. Fewer iterations, a weaker trust region or a reduced learning rate can help to preserve more diversity, while trading off some of the final performance.



Figure 13: **Design search trajectories for the Unitree Go2 quadruped.** The columns show 8 different initial designs, and the rows show the initial design and VGDS at iteration 10, 20, 30, 40, and 50.

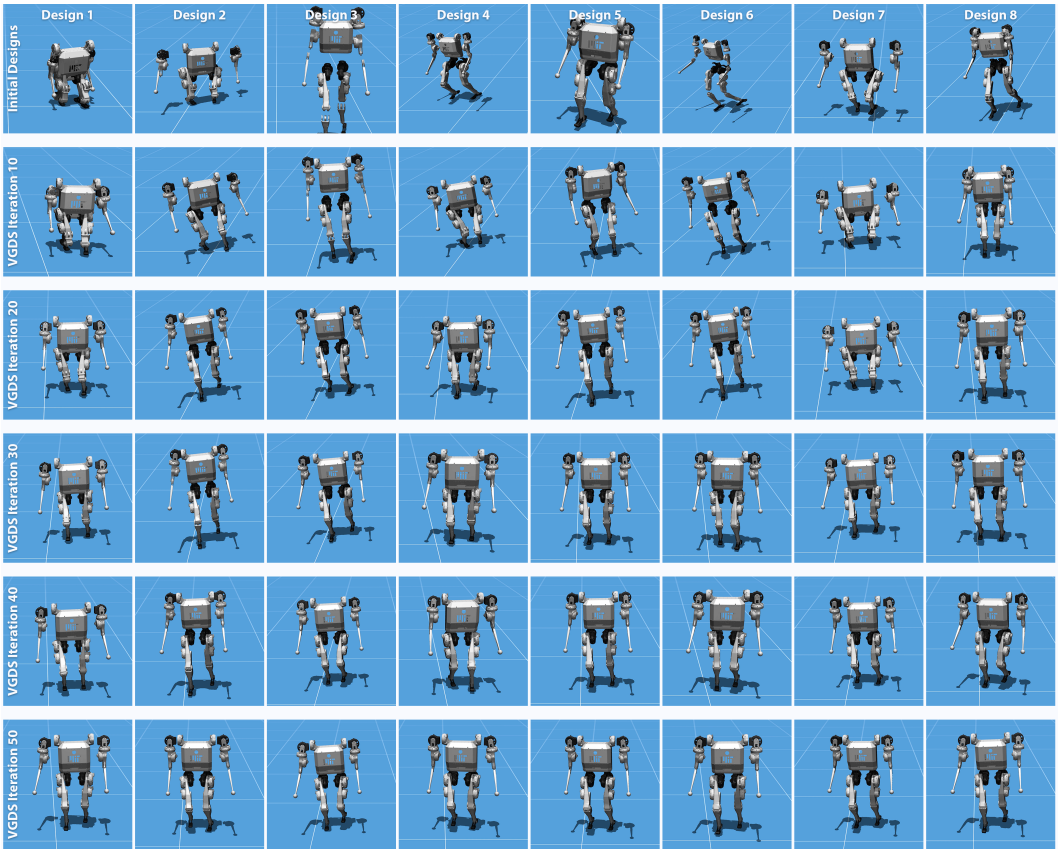


Figure 14: **Design search trajectories for the MIT Humanoid.** The columns show 8 different initial designs, and the rows show the initial design and VGDS at iteration 10, 20, 30, 40, and 50.



Figure 15: **Design search trajectories for the Golem hexapod.** The columns show 8 different initial designs, and the rows show the initial design and VGDS at iteration 10, 20, 30, 40, and 50.

J Additional Method Variants

During the development of VGDS, we tested several variants for the search, and while they did not outperform the final method, they provided useful insights into the design choices and failure modes. We hope that reporting these negative results can help future work, so we summarize the most informative variants here.

Hard trust-region projection. We tested a hard trust region, where the design is projected back into a fixed-radius ball around the reference design after each Adam step. This proved to be competitive with the soft L2 objective in most settings. However, it introduces an additional radius that must be chosen explicitly, and we found that tuning this radius was slightly more difficult than tuning the smoother L2 penalty coefficient λ.

Uncertainty-based trust regions. A natural alternative is to use disagreement between critic heads as a trust region. We tried absolute and relative uncertainty formulations, as well as lower confidence bound objectives. While these versions could get rid of the dependence on a reference design, they did not provide a reliable trust region in the end. We found that the small ensemble of value heads, trained on the same data, could still agree on some catastrophic extrapolations.

Search dynamics. We tested several optimizer modifications, including initial design noise, per-step gradient noise, Adam moment resets, L2-norm update clipping, cosine learning rate schedules, SGD instead of Adam, and sweeps over learning rate, step size, and number of iterations. None of these changes produced consistent improvements over the simple Adam configuration that we ended up using. More search iterations can sometimes even hurt when over-optimizing the critic.

Multi-start search. We also tested a multi-start version, where multiple particles are initialized with small perturbations, moved through the gradients, and the best particle is selected by the surrogate objective. We found that once the trust region objective is chosen well, multi-start search did not lead to consistent improvements over the simpler single-design version.

Direction and subspace structure. We tested whether the useful design update is concentrated in a small number of dimensions. Active subspace variants that restricted updates to the largest gradient dimensions degraded performance. Similarly, removing one dominant search direction was mostly harmless, but restricting the update to only that direction collapsed the performance. This showed us that the useful design signal is mostly dense and high-dimensional.

Refreshing the state bank. One natural concern of the fixed state bank idea is that it becomes inconsistent as the design changes during search. We therefore tried refreshing the states by rolling out the policy under the current design during optimization or sampling states from the starting-state distribution of the current design. This did not improve results and was on rare occasions even worse. We hypothesize that the fixed state bank works because the generalist policy is robust enough that the state distributions across closely related embodiments heavily overlap, and the trust region keeps the new design within that valid neighborhood.