Transformer Transformer: A Unified Model for Motion-Conditioned Robot Co-design

Huy Ha[1,2], Karen Liu[1], Shuran Song[1,2]

[1]Stanford University [2]Columbia University

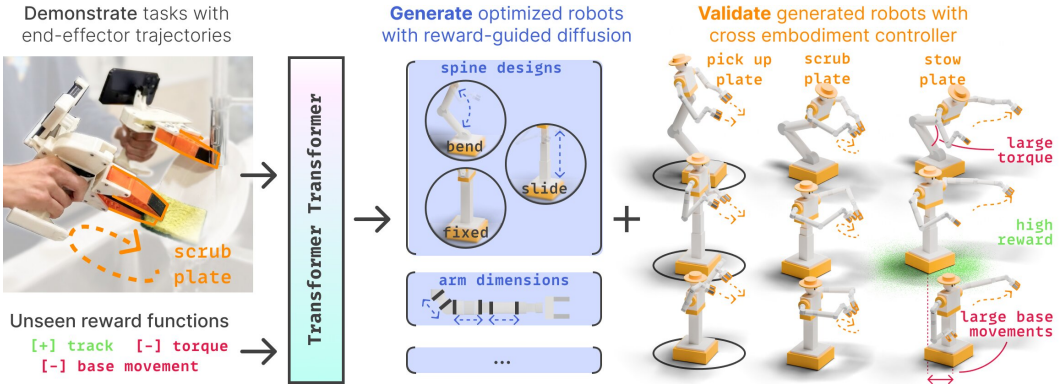

Figure 1: **Demonstrate, Generate, Validate.** From target end-effector motions and *unseen* reward functions, Transformer Transformer **generates optimized robot** designs that maximize rewards, then **validates generated robots** by controlling them to track the given motions. Trained over robot embodiment, state, and action tokens, it unifies embodiment optimization and cross-embodiment control into one model, providing a one-stop shop for robot co-design from human demonstrations.

Abstract: An often overlooked factor of robot manipulation performance is the embodiment of the robot itself. Motivated by this problem, we study motion-conditioned robot co-design, where the goal is to generate complete robot designs that track target end-effector trajectories (from human demonstrations) while optimizing user-defined rewards. We introduce Transformer Transformer, a diffusion transformer trained on RoboTokens, a unified tokenization of robot embodiments, states, and actions. The same architecture can be used across embodiment spaces (e.g., wheeled bimanual, quadrupeds, humanoids) and use cases (embodiment generation, cross embodiment controller). Rather than overfitting to one reward function, Transformer Transformer is a dynamics model, whose reward-agnostic state and action predictions can be converted into reward-specific value predictions. These value predictions are used to steer embodiment diffusion towards high value robot designs, through a procedure we call Dynamics Self-Guidance. Experiments across multiple design spaces show zero-shot optimization of unseen rewards and trajectories, improving performance and runtime over the evolutionary baseline. Finally, we fabricated an optimized ALOHA design, which reduced tracking error by over 70% compared to the original design. Check out transformer-transformer.github.io for summary/result videos.

Keywords: Generative Robot Co-design, Cross-Embodiment Control

1 Introduction

An often overlooked factor of manipulation performance is the embodiment of the robot itself. What motions robots can effectively perform depends on its embodiment. Motivated by this, we ask:

What is the best robot embodiment for a given manipulation task?

We formalize manipulation tasks as end-effector trajectories, a widely adopted task representation for manipulation policy learning [1–13]. While these end-effector trajectories are embodiment-agnostic, their execution is not. Instead, it critically depends on the tracking performance dictated

by the robot embodiment. As a result, policies trained on such data transfer unequally to different robots [13]. Seen this way, imperfect transfer is not only a limitation—it is also an opportunity.

In this work, we study *motion-conditioned robot co-design*. Given (i) a set of target end-effector trajectories and (ii) a set of user-defined reward functions, our goal is to automatically generate *complete* robot embodiments that achieve high rewards (Fig. 1). Here, a complete robot embodiment means reasoning over large, heterogeneous design spaces spanning kinematics, geometry, and dynamics parameters. Further, the user-specified rewards can be functions of both the robot's embodiment (e.g., size, mass) and dynamic behavior (e.g., joint velocity, tracking precision).

To tackle this problem, we propose **Transformer Transformer**[1], a DiT [14] designed to be a one-stop shop for motion-conditioned robot co-design, enabled by the following contributions:

- **Unified Robot Representation:** We propose RoboTokens, a tokenization scheme that can represent any articulated robot's embodiment, states, and actions. Designed to flexibly store complete robot data in a consistent format, RoboTokens provides the foundational representational capability that enables a single model to span diverse robot embodiment spaces.
- **Unified Architecture:** Trained with different masked modeling schemes over RoboTokens, our model acts simultaneously as the generator, critic, and controller for co-design problems. By consolidating all co-design modules into a single neural network, our optimization pipeline is simple and readily GPU-parallelized.
- **Unified Dynamics Training Objective:** Instead of specializing on a reward-specific signal, our network is trained to model the general dynamics of diverse robot embodiments. To zero-shot optimize for user-specified reward functions at inference time, we convert our model's *reward-agnostic* dynamics prediction into a *reward-specific* score prediction. This predicted score is used to guide the model's embodiment diffusion process to produce high-value embodiments.

Across three robot design spaces (i.e., fixed-base, quadruped, and bimanual mobile manipulator), we demonstrate that Transformer Transformer can zero-shot optimize user-specified end-effector trajectories and reward functions at inference time. Conditioned on the robot embodiment, the same model can also be used as a cross-embodiment whole-body controller, directly usable for design validation. Finally, we validate our model's predictions by fabricating an optimized ALOHA design on bimanual flinging for cloth unfolding, reducing tracking error by 73% and maximum joint velocity by 30%. Check out the transformer-transformer.github.io for more results.

2 Method

Overview: The overarching goal of Transformer Transformer is to model the joint distribution of robots' embodiment and dynamics (i.e., states and actions). To enable this goal, we first describe RoboToken (§ 2.1), a tokenization scheme that can represent the embodiments and dynamics from diverse robots. Next, we will describe how to train a diffusion transformer over these tokens (§ 2.2), along with its applications in motion-to-robot generation and cross-embodiment control policies. To enable more efficient sampling, we describe dynamics self-guidance, a method for guiding the diffusion process to optimize unseen reward functions without training any additional network (§ 2.3).

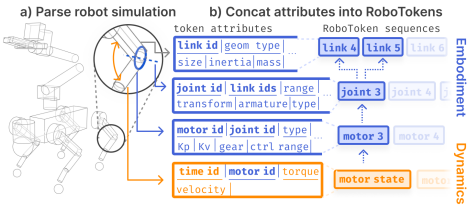

Figure 2: **A Unified Robot Representation**. RoboToken is designed to be a complete, learning-ready token representation of rigid articulated robots. Our tokenizer can convert any robot description into a sequence of RoboTokens, consisting of time-invariant **embodiment** tokens and time-varying **dynamics** tokens.

Scope: The RoboToken representation was designed to represent rigid articulated robots (same scope as MJCF) optimized for robot learning. It is currently limited to primitive-based geometry representation. Further, we focus on end-effector trajectories from UMI [1] demonstrations as the manipulation task representation, which is widely adopted [1–12]. Finally, we aim to design a framework that can generalize to reward functions over the robot's embodiment, states, and controls. Rewards involving other properties (e.g., structural strength, appearance) are out of scope.

2.1 RoboToken: A Unifying Robot Representation

Complete. RoboToken includes all attributes of a robot's embodiment (time-invariant) and dynamics (time-varying). The tokenizer converts any robot description into a set of five embodiment token

[1]The first "Transformer" refers to shape-shifting robots; the second refers to the self-attention architecture.

types: links, fixed joints, sliding/rotating joints, ball joints, and motors (Fig. 2, only three embodiment types shown for illustration purposes). In addition, the tokenizer also converts each episode of states and actions into a set of four state token types (all but fixed joints), along with action tokens. For instance, each link token contains the physical parameters of the link (primitive type, primitive size, inertia), while link state tokens include the link's pose at a given timestep. Within each token type, all data attributes are concatenated into a learning-ready, continuous-valued vector.

Flexible. To encompass diverse robot kinematics, RoboToken supports a variable number of embodiment tokens representing arbitrary connectivity. Here, joints point to the two links they connect via two link IDs, while motors point to the joint they control with joint IDs (Fig. 2, dotted arrows). This supports passive joints when no motors point to them (e.g., Cassie's leg mechanism, Fig. 3). To point to each other, tokens include each other's IDs in their data attributes. To support heterogeneous state and action spaces, we represent each link/joint/motor state at each timestep as its own token, pointing to the corresponding timestep and embodiment token. These IDs are used later in training to learn positional embeddings for each ID type (Fig. 4).

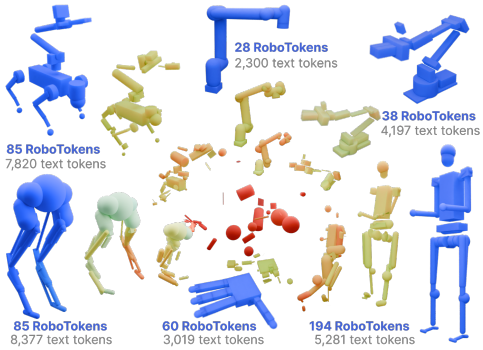

Figure 3: **RoboToken Diffusion**. Transformer Transformer diffuses noise into RoboTokens (token count in blue), 27-110× more compact than MJCF text tokens (token count in grey).

Consistent. Robot description formats like MJCF do not have strict conventions regarding transform handling, opting for ease of human editing instead of data representation consistency. As such, directly learning over these text representations would force networks to learn redundant spatial offsets that are equivalent, introducing extra variance without extra information. The RoboToken tokenizer enforces consistent transform conventions for representing geometry and dynamics information, automatically splitting inertias, and collapsing transforms in the preprocessing stage.

Extensible. Our implementation of RoboToken makes it frictionless to support extra tokenization/detokenization logic for different robot tasks, embodiment, and dynamics data. To illustrate this, we adapted RoboToken for the trajectory tracking task by adding target end-effector pose tokens to the RoboToken format (Fig. 4, "Target Pose" tokens). These conditioning tokens aren't noised during training, and point to the end-effector ID that should track it and the timestep ID.

Optimizability. While text-based formats like MJCF can be used for generation, they are difficult to *optimize* over. LLMs are autoregressive, categorical models that lack *global controllability* (i.e., later tokens can't affect earlier ones) and *differentiability* (due to sampling and detokenization). In contrast, diffusing continuous-valued RoboToken has both these properties.

2.2 Transformer Transformer: A Unifying Architecture

To train a model over RoboTokens, we need an architecture that can model *variable* sequence lengths of *high-precision continuous-valued* data. To achieve this, we use a diffusion transformer (DiT)-based architecture [14] (Fig. 4). Trained with different masked modeling schemes, our model can both generate robot designs and control them for motion-conditioned co-design, as follows:

Motion-to-Robot Optimization. Given target motions and reward functions, the model must generate high-value robot designs. Evaluating these reward functions often requires both information about the embodiment (e.g., mass penalty) and dynamics (e.g., torque penalty) while tracking the target motion (e.g., tracking reward). Thus, conditioned on the target motion trajectory, our model *simultaneously* diffuses embodiment and dynamics tokens, which gives the model everything it needs to assess newly generated designs. Beyond speed, this non-autoregressive formulation also avoids the error accumulation inherent to autoregressive dynamics models, yielding more temporally coherent long-horizon predictions for design evaluation. When trained on data from diverse robots, the model must learn long-range dependencies between robot design, states, and controls over extended time horizons. Since the model also outputs action tokens, this optimization procedure also considers control. We construct an embodiment generator biased toward high-value candidates by running n parallel diffusion processes, using the specified reward function to rank the n candidates, and returning the best one. We refer to this embodiment optimizer as the **Zeroth-Order Optimizer**.

Cross-Embodiment Control. We formulate a cross-embodiment controller that is embodiment-aware: conditioning on the embodiment [15–21], current state, and target motion, it predicts the

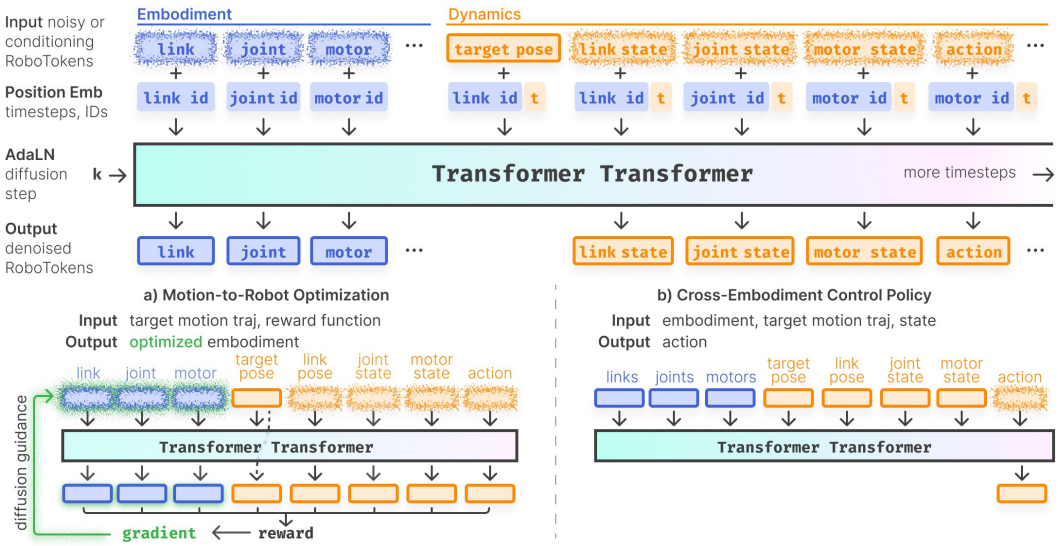

Figure 4: **Unified Architecture for Robot Co-Design**. Transformer Transformer is a diffusion transformer, trained to jointly diffuse **embodiment** and **dynamics** RoboTokens, where different conditioning in the same model enables (a) motion-to-robot optimization and (b) cross-embodiment control. At inference time, the model uses unseen rewards to turn its **embodiment** and **dynamics** predictions into a **reward** prediction, and incorporates its **gradients** back into the diffusion process.

expert actions that were used to track that motion. Due to RoboToken's completeness, this conditioning provides critical information about the robot's kinematics, dynamics, and actuation capabilities to enable embodiment-aware action predictions. Due to RoboToken's flexibility, heterogeneous state and action spaces (e.g., due to different DoF count) can be handled by varying the number of state and action tokens in the model's context. When trained on data from diverse robots, the model can generalize to unseen embodiments at inference time.

Model & Training. For each token, we learn a token-type-specific linear projection from that type's vector space into the DiT's latent space, and add learned embeddings of the token's IDs. Since our domain exhibits $\mathrm{SE}(2)$ equivariance (not $\mathrm{SE}(3)$, due to gravity), we apply planar transform augmentations. Each token type's sequence is padded to the same maximum sequence length, then all token type sequences are concatenated together to form one long, multi-modal sequence. We train the DiT using the DDIM scheduler [22]. To speed up training, we subsample episode timesteps to form shorter, fixed-length sequences, and found that 8 timesteps were sufficient for the model to be effective for our use cases. These timesteps are randomly sampled within an episode for the motion-to-robot task, and consecutively sampled for the control task to give it past action/future target information. Hyperparameters are reported in Tab. 20 and more model details in § 9.2.

2.3 Optimization with Dynamics Self-Guidance

In challenging optimization landscapes, zeroth-order optimizers may take many samples to return high value designs. To address this, we introduce Dynamics Self-Guidance, a first-order diffusion guidance method that uses gradients from the model's noisy reward predictions to optimize its noisy embodiment predictions. While prior works have explored converting dynamics into diffusion guidance signals, they require training a separate dynamics model [23] or a differentiable simulator [24]. Meanwhile, our approach uses a single, unified diffusion model as follows.

Given a differentiable reward, we can backpropagate its gradient to the embodiment tokens. At step k, the DiT predicts ϵ_k, which is used to denoise tokens from step k to step $k-1$. Thus, even if the full diffusion process is not differentiable, per-step reward gradients can still guide samples toward higher-reward candidates. We incorporate this gradient information following classifier-guided DDIM [25]. At inference time, we run n parallel guided diffusion processes, then return the best design ranked by the model. We call this embodiment optimizer **Dynamics Self-Guidance**.

2.4 Data Collection & Generation

Target Motion Dataset. We collected 76 motion trajectories using an extension of the UMI gripper [1, 13, 26], which is built on top of ARKit (Fig. 1, left). These target trajectories include single end-effector motions for diverse skills like tossing, screwing, opening drawers, and scrubbing. We

use 56 trajectories for training, and 20 trajectories for validation. In addition, we use UMI's bimanual dishwashing demonstrations for the bimanual design space.

Design Spaces. We picked three design spaces to stress-test three co-design challenges:

- **Kinematic Design**: The Trossen ViperX 300S design space (Fig. 6, right), varies mounting pose, degrees of freedom count, joint orientations, and link lengths. While fixed-base arms simplify control, they are unforgiving to poor kinematic design. Failure to reason about kinematic reachability will lead to large tracking penalties.
- **Dynamic Control**: The quadruped manipulator design space (Fig. 6, left) includes leg design variations (e.g., serial vs. spring-loaded linkage), arm mounts (fixed vs. sliding rail), on top of length variations for torso, legs, and arms. To successfully design high-value quadruped manipulators, the algorithm must reason through each design's whole-body control performance, paying attention to how stability and inertia trade off with agility for precise, dynamic tracking.
- **Task Complexity**: The mobile bimanual design space (Fig. 1) includes different spine designs (fixed, sliding, bending) on top of length variations. Bimanual, whole-body manipulation encapsulates tasks more complex than those involving only a single end-effector, including long-horizon, coordinated movements with many subtasks like in dishwashing.

Our design spaces are combinatorial, with discrete and continuous choices, providing diverse RoboTokens. Next, we discuss how all generated robots are controlled to track target motions, completing our training dataset with state and action RoboTokens.

DiffIK Control for Non-legged Robots. For the fixed-base and mobile bimanual design spaces above, we use Mink [27], a differential inverse kinematics library for MuJoCo [28]. Since this kinematic solver doesn't consider dynamics, we account for joint position control lag with a fixed look-ahead time of 80 ms. We include terms for tracking, posture regularization, and damping.

Whole-body Control for Legged Robots. Reinforcement learning (RL) [29, 30] is the dominant paradigm for whole-body control of legged robots [31–36]. However, training one policy per each robot [15, 20] is too costly. Thus, we train one policy for each discrete variation (e.g., leg design), while including continuous variations (e.g., leg lengths) in the policy observation, allowing one policy to control many continuous variations. We extend manipulation-centric WBCs [31] to include continuous variation observations. For each combination of 7 binary design choices, we train an RL policy used for data collection and design validation, yielding 128 RL experts in total. We also investigated GPU-accelerated trajectory optimization [37], but found that RL controller training was faster in total wall clock time for data generation. We report more data generation details in § 9.3.

3 Results

RoboToken Unifies Diverse Robot Embodiment Spaces. We tokenized eleven robots from MuJoCo Menagerie [38], including a dexterous hand, fixed-base arms, quadrupeds (+ manipulators), bipeds, and humanoids. Their masses span two orders of magnitude (Allegro V3's 0.65 kg to Anymal C's 67.5 kg), with DoF counts ranging from 6 active joints (UR5e) to 35 active/passive joints (Cassie), but all tokenize into RoboToken sequences of lengths 28 to 101.

Generated Designs Lie on the Training Manifold. A natural question is what range of designs Transformer Transformer can produce. Empirically, our generator inherits properties theoretically expected of diffusion models [39, 40]. *Manifold adherence*: since the score function is undefined off the training distribution, the learned denoiser steers samples toward the data manifold rather than beyond it. Generated designs interpolate within the training distribution but do not extrapolate beyond it—e.g., the model cannot produce hexapods from quadruped-only training, nor link lengths exceeding the training range. This is both a feature (physically plausible outputs by construction) and a limitation (extending the design space requires extending the dataset). *Mode coverage*: an unconditional model generates each robot type with probability matching the training distribution. *Cross-attribute coherence*: generated outputs respect the joint distribution of embodiment attributes—larger links carry proportional masses and inertias, motors are sized to the joints they actuate—ensuring physical feasibility without explicit constraints.

Transformer Transformer Optimizes Embodiments for Unseen Reward Functions Zero-Shot. At inference time, we optimize the generated robot hardware to maximize *unseen* reward functions, conditioned on *unseen* target tracking motions. These rewards include terms for embodiment (e.g., size, weight) as well as behavior while tracking the target trajectories (e.g., tracking performance, torque/velocity usage). We include reward definitions and additional metrics in the appendix (§ 8).

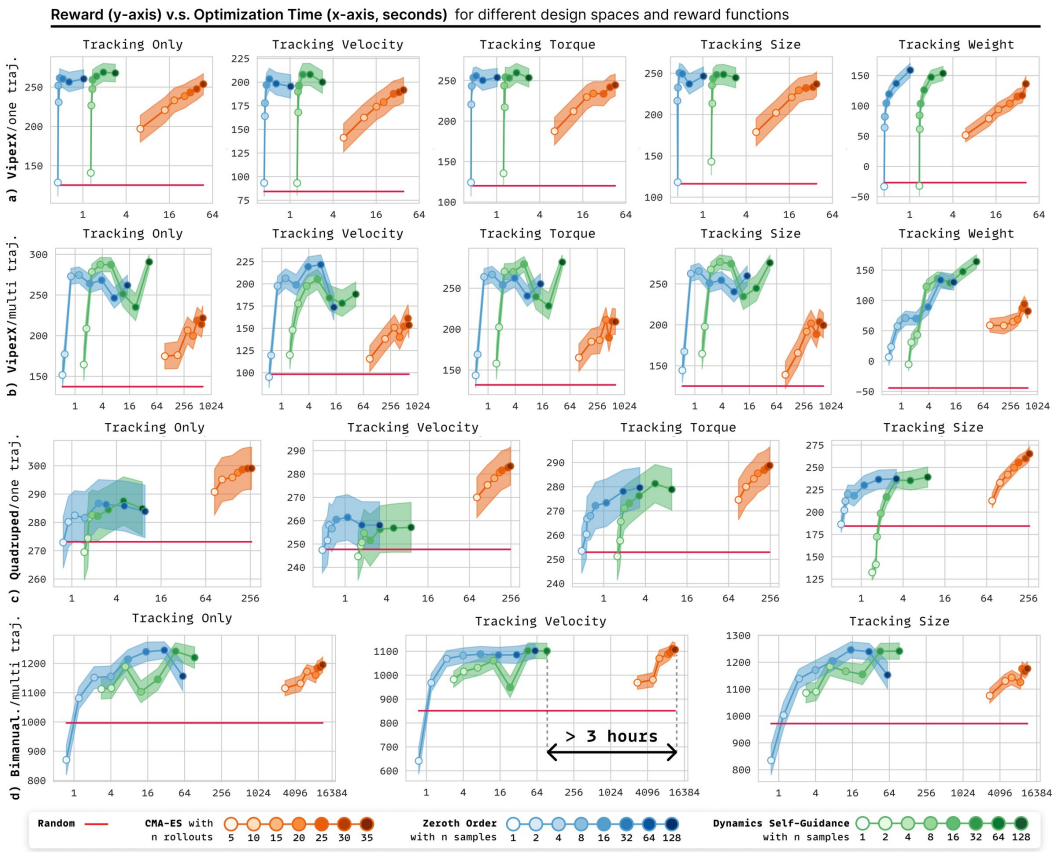

Figure 5: **Better Designs with Test-Time Scaling**. Across three design spaces and all rewards (§ 8), both our **Zeroth Order** and **Dynamics Self-Guidance** tend to generate higher-reward designs when allowed more parallel samples. Our model also zero-shot optimizes multiple trajectories (b,d).Our approach is significantly faster than **CMA-ES**, achieving similar or better performance while taking orders of magnitude less time.

Approaches. (1) **Random** samples the discrete and continuous choices; (2) **CMA-ES** [41, 42], a widely applied evolutionary algorithm in co-design [43–45], samples the discrete choices and then optimizes the continuous choices; (3) **Zeroth Order** samples n robots from our model, ranks them with the model, and returns the highest ranked one ; (4) **Dynamics Self-Guidance** extends Zeroth Order by introducing gradient guidance. To isolate embodiment optimization performance in our approaches, we train a model for each design space, and on only the motion-to-robot task.

Evaluation Procedure. To account for optimization efficiency, we report wall-clock time taken for the optimization procedure to produce the optimized robot. For each optimized robot, we validate the design in MuJoCo [28] with the design space's controller (RL experts/Mink § 2.4), and report the average sum of rewards over the episode. In Fig. 5, we show the mean with 95% confidence interval bars over held-out trajectories and CMA-ES/diffusion sampling 9 seeds. All time measurements were conducted on a lightly loaded NVIDIA RTX 4090 and an Intel i9 processor core.

Parallelized Design Evaluation. For the fixed-base and bimanual mobile design spaces (a,b,d), our model outperforms CMA-ES in terms of both speed and performance (Fig. 5). Beyond the advantages of GPU parallelization along the batch dimension of zeroth-order search, our approach enjoys a second axis of parallelization, along an episode's time dimension. For CMA-ES, evaluating a candidate design requires running the controller and simulator over thousands of timesteps *sequentially*. In contrast, our model can reason about long-horizon dynamics (and thus performance) by considering a small set of episode timesteps *in parallel*, enabled by our non-autoregressive formulation. These two factors of parallelization multiply, allowing our approach to diffuse and evaluate 128 candidates much faster than CMA-ES can evaluate 5 candidates.

Diffusion Generates Diverse Designs. To accurately track the tossing trajectory, our model generally prefers larger robots (more stable) with longer arms (faster velocity) (Fig. 6, middle left). However, when the optimization landscape shifts to regularize the robot's size, our model converges

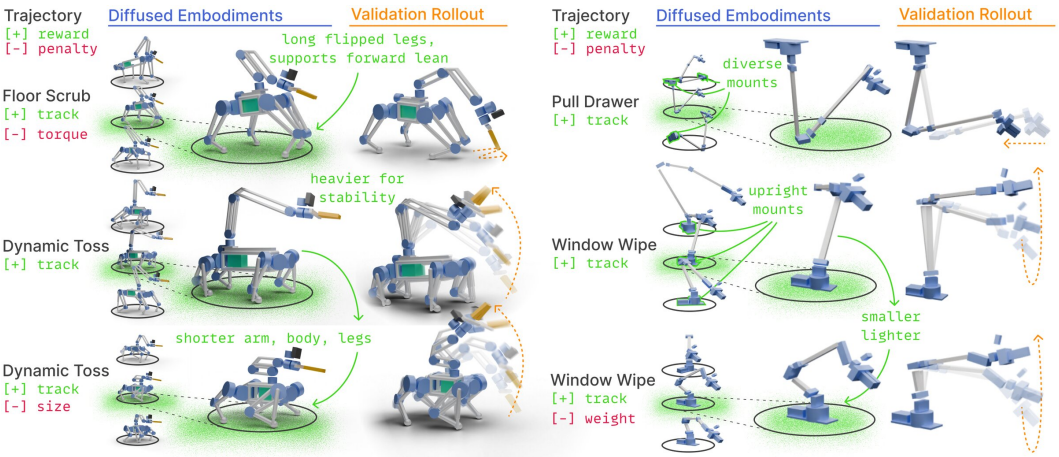

Figure 6: **Diffusion Generates Diverse Designs**. Our model diffuses embodiments with discrete (e.g., leg designs, degrees of freedom) and continuous (e.g., arm length, mounting points) variations that maximizes the specified reward functions. Changing the target trajectory (e.g., "Floor Scrub" → "Dynamic Toss") or the reward (e.g., "size" penalization) changes the optimization landscape, leading to different embodiment distributions (e.g., diverse → only upright mounts).

on smaller robots. Our model's ability to produce diverse robots for different reward functions arises from RoboToken's completeness, which gives reward functions many handles to control the diffusion process. Meanwhile, the model's ability to produce diverse robots for the same reward function arises from using a diffusion formulation for Transformer Transformer.

Diffusion Composition for Multi-Trajectory Optimization. We apply diffusion composition [46, 47] to enable zero-shot multi-trajectory optimization from only single trajectory training. Running the composed diffusion process generates designs that perform favorably over many unseen trajectories simultaneously (20/26 trajs in Fig. 5b,d). Our approach parallelizes favorably, while CMA-ES's optimization time scales linearly with the number of trajectories, taking *3 hours longer*. Unlike many computational design prior works which focus on task-specific designs [23, 44, 48–52], our ability to optimize for multiple trajectories broadens its potential applications, such as in optimizing a generalist manipulation platform by conditioning on a motions from different tasks from a manipulation dataset [53].

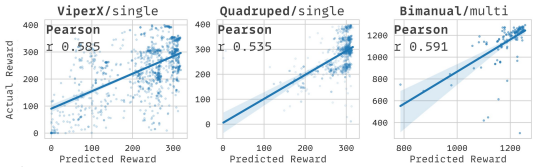

Figure 7: **Predicted Reward Accuracy**. Learning legged robot dynamics means reasoning about how learned whole-body controllers navigate discontinuous contacts and termination risks from falling over. Thus, we observe higher correlation between predicted and actual rewards for ViperX and Bimanual than Quadruped design spaces.

Better Designs with Test-Time Compute. When given more test-time compute, language reasoning models generate better outputs [54–56]. We observe a similar phenomenon in our model (Fig. 5), where increasing test-time compute by searching over more parallel seeds [57] allows the model to produce better designs. Notably, this behavior holds across all design spaces and reward functions, emerging purely from learning to model robot embodiments and dynamics. Although our model can refine its embodiment design for up to a minute (Fig. 5d, guided), performance does not robustly increase with test-time compute, as observed in earlier language models [57–59]. Future work could investigate improving dynamics (and thus reward) modeling accuracy (Fig. 7) and alternative inference schemes [57, 59] to improve test-time compute scaling.

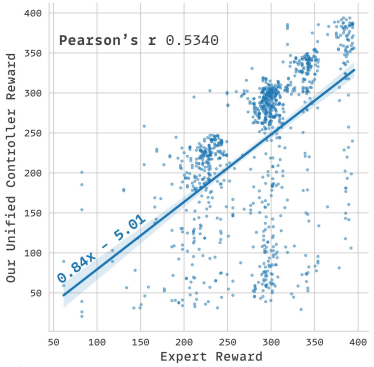

Figure 8: **Whole-body Control of Generated Quadrupeds**. After generating optimized robots, the same model can directly control its designs.

Transformer Transformer Controls Unseen Robots. We study our model's ability to *faithfully* self-validate designs through controlling its generated embodiments. Here, faithful means embodiments should achieve similar reward between experts and self-validation. To perform well, the learned controller must transfer to diffused embodiments despite being trained on clean, procedurally generated robots. Failure to do so may lead to the robot

falling over, truncating the episode and the sum of rewards. Despite this challenge, we observe a positive correlation (Pearson's r: 0.53, Fig. 8). Although there are some outlier embodiments, most cluster along the diagonal, demonstrating promising first steps towards replacing the 128 RL experts with a single, unified cross-embodiment controller. We report more results in the appendix (Fig. 11).

Real-world validation. We fabricated an ALOHA setup [60, 61] optimized by our model for the "Tracking Velocity" reward (Fig. 9,10). We focus on flinging for cloth unfolding [62], which stress-tests the framework with challenging dynamic manipulation and demands robustness to modeling errors and unmodeled disturbances (e.g., aerodynamic drag, cloth friction and weight). Compared to the original design, the optimized design achieved a 30% reduction in maximum joint speed ($2.57 \rightarrow 1.82$ rad/s) and a 73% reduction in tracking position error ($13.0 \rightarrow 3.5$ cm). The optimized design featured link lengths long enough to track the full motion while remaining light enough to be supported by ALOHA's Dynamixels motors. It also mounted the arms upside down behind the workspace, enabling a more efficient underarm swing instead of an overhead fling.

4 Related Works

Embodiment Design and Optimization. Traditional gradient-based methods require limited embodiment representations (e.g., cage-based [44, 50]), heuristics [48, 63], and don't extend to domains with complex control and contacts [63–66] (e.g., RL for legged robots). Data-driven methods are flexible but overfit to one reward [49, 51]. Dynamics-guided generation [23, 52] generalizes across rewards with a dynamics model whose gradients steer design, but still opts for fixed open-loop [23, 49] or simplified kinematic actions [48, 67], sidestepping the full control problem. Transformer Transformer instead learns a dynamics model over any action space for design.

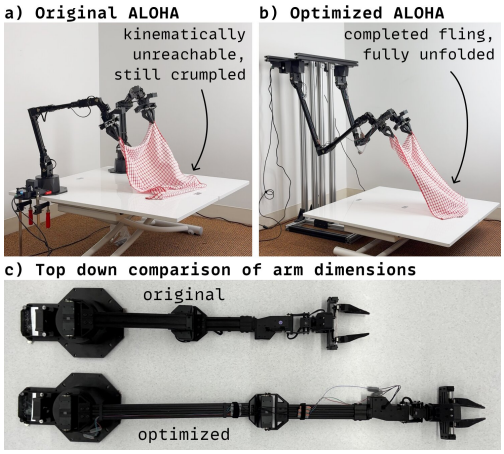

Figure 9: **Dynamic Cloth Unfolding.** To optimize ALOHA for high speed flings, our model found better mounting point and link dimensions, reducing tracking error by 73% and max joint speed by 30%.

Cross-Embodiment Control. GPU-based simulators [68, 69] have made RL [29, 30] the dominant approach for locomotion and whole-body control [31–36, 68, 70–73], with cross-embodiment extensions by distilling single-embodiment experts into a shared policy [15, 20] that conditions on the embodiment [15–21]. However, their embodiment representations are typically incomplete.

Robot Co-design. Evolutionary algorithms [74–78] showed that, given a simulator, enough compute will find high-value designs. Subsequent data-driven work accelerates this loop with learned design generators [16, 19, 24, 79–82], critics [82–85], or controllers [16, 19, 21, 43, 79, 82, 85], but these components are trained on disjoint representations and objectives, with iterative pipelines that are hard to speed up. Transformer Transformer instead unifies generator, critic, and controller into one model, consolidating the co-design process into a single GPU-accelerated diffusion process.

5 Conclusion

We proposed Transformer Transformer, a DiT that performs generative robot design and control, enabling a one-stop shop for motion-conditioned robot co-design. Our simulation results across three designs spanning fixed-base, legged, and bimanual mobile manipulation demonstrated the model's efficiency, performance, and zero-shot capabilities to unseen rewards, and our real world results validated the model's robustness to misspecified/unmodelled dynamics and practical usage.

6 Limitations

Towards a more complete co-design formulation for manipulation, RoboToken's scope should be expanded to support complex geometry [23, 24, 44, 49, 50], scene/object [86], and tactile information [26]. Further, scaling RoboToken data generation (e.g., more efficient legged control) to more diverse robots will be critical towards a foundation model that reasons about the robot's embodiment as first-class tokens. Ultimately, such advances would allow us to explore robot morphologies as diverse as the manipulation tasks they perform, moving beyond robot hardware as a static constraint.

References

[1] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[2] Z. Liu, C. Chi, E. Cousineau, N. Kuppuswamy, B. Burchfiel, and S. Song. Maniwav: Learning robot manipulation from in-the-wild audio-visual data. *arXiv preprint arXiv:2406.19464*, 2024.

[3] F. Lin, Y. Hu, P. Sheng, C. Wen, J. You, and Y. Gao. Data scaling laws in imitation learning for robotic manipulation. *arXiv preprint arXiv:2410.18647*, 2024.

[4] Z. Wu, T. Wang, C. Guan, Z. Jia, S. Liang, H. Song, D. Qu, D. Wang, Z. Wang, N. Cao, et al. Fast-umi: A scalable and hardware-independent universal manipulation interface. *arXiv preprint arXiv:2409.19499*, 2024.

[5] M. Seo, H. A. Park, S. Yuan, Y. Zhu, , and L. Sentis. Legato: Cross-embodiment imitation using a grasping tool. *IEEE Robotics and Automation Letters (RA-L)*, 2025.

[6] F. Liu, C. Li, Y. Qin, A. Shaw, J. Xu, P. Abbeel, and R. Chen. Vitamin: Learning contact-rich tasks through robot-free visuo-tactile manipulation interface. *arXiv preprint arXiv:2504.06156*, 2025.

[7] M. Xu, H. Zhang, Y. Hou, Z. Xu, L. Fan, M. Veloso, and S. Song. Dexumi: Using human hand as the universal manipulation interface for dexterous manipulation. *arXiv preprint arXiv:2505.21864*, 2025.

[8] T. Tao, M. K. Srirama, J. J. Liu, K. Shaw, and D. Pathak. Dexwild: Dexterous human interactions for in-the-wild robot policies. *Robotics: Science and Systems (RSS)*, 2025.

[9] X. Zhu, B. Huang, and Y. Li. Touch in the wild: Learning fine-grained manipulation with a portable visuo-tactile gripper. *arXiv preprint arXiv:2507.15062*, 2025.

[10] G. Lee, Y. Lee, K. Kim, S. Lee, S. Noh, S. Back, and K. Lee. Manipforce: Force-guided policy learning with frequency-aware representation for contact-rich manipulation. *arXiv preprint arXiv:2509.19047*, 2025.

[11] O. Rayyan, J. Abanes, M. Hafez, A. Tzes, and F. Abu-Dakka. Mv-umi: A scalable multi-view interface for cross-embodiment learning. *arXiv preprint arXiv:2509.18757*, 2025.

[12] Y. Xu, L. Wei, P. An, Q. Zhang, and Y.-L. Li. exumi: Extensible robot teaching system with action-aware task-agnostic tactile representation. In *Conference on Robot Learning*, pages 2536–2554. PMLR, 2025.

[13] H. Gupta, X. Guo, H. Ha, C. Pan, M. Cao, D. Lee, S. Sherer, S. Song, and G. Shi. Umi-on-air: Embodiment-aware guidance for embodiment-agnostic visuomotor policies, 2025. URL https://arxiv.org/abs/2510.02614.

[14] W. Peebles and S. Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

[15] H. Furuta, Y. Iwasawa, Y. Matsuo, and S. S. Gu. A system for morphology-task generalization via unified representation and behavior distillation. *arXiv preprint arXiv:2211.14296*, 2022.

[16] 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.

[17] V. Kurin, M. Igl, T. Rocktäschel, W. Boehmer, and S. Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*, 2020.

[18] A. Gupta, L. Fan, S. Ganguli, and L. Fei-Fei. Metamorph: Learning universal controllers with transformers. *arXiv preprint arXiv:2203.11931*, 2022.

[19] H. Lu, Z. Wu, J. Xing, J. Li, R. Li, Z. Li, and Y. Shi. Bodygen: Advancing towards efficient embodiment co-design. *arXiv preprint arXiv:2503.00533*, 2025.

[20] 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.

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

[22] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*.

[23] X. Xu, H. Ha, and S. Song. Dynamics-guided diffusion model for robot manipulator design. *arXiv preprint arXiv:2402.15038*, 2024.

[24] T.-H. J. Wang, J. Zheng, P. Ma, Y. Du, B. Kim, A. Spielberg, J. Tenenbaum, C. Gan, and D. Rus. Diffusebot: Breeding soft robots with physics-augmented generative diffusion models. *Advances in Neural Information Processing Systems*, 36:44398–44423, 2023.

[25] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[26] H. Choi, Y. Hou, C. Pan, S. Hong, A. Patel, X. Xu, M. R. Cutkosky, and S. Song. In-the-wild compliant manipulation with umi-ft. *arXiv preprint arXiv:2601.09988*, 2026.

[27] K. Zakka. Mink: Python inverse kinematics based on MuJoCo, Dec. 2025. URL https://github.com/kevinzakka/mink.

[28] 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.

[29] R. S. Sutton, A. G. Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

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

[31] H. Ha, Y. Gao, Z. Fu, J. Tan, and S. Song. UMI on legs: Making manipulation policies mobile with manipulation-centric whole-body controllers. In *Proceedings of the 2024 Conference on Robot Learning*, 2024.

[32] Z. Fu, X. Cheng, and D. Pathak. Deep whole-body control: learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning*, pages 138–149. PMLR, 2023.

[33] M. Ji, X. Peng, F. Liu, J. Li, G. Yang, X. Cheng, and X. Wang. Exbody2: Advanced expressive humanoid whole-body control. *arXiv preprint arXiv:2412.13196*, 2024.

[34] M. Liu, Z. Chen, X. Cheng, Y. Ji, R.-Z. Qiu, R. Yang, and X. Wang. Visual whole-body control for legged loco-manipulation. *arXiv preprint arXiv:2403.16967*, 2024.

[35] T. Portela, A. Cramariuc, M. Mittal, and M. Hutter. Whole-body end-effector pose tracking. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11205–11211. IEEE, 2025.

[36] T. Portela, G. B. Margolis, Y. Ji, and P. Agrawal. Learning force control for legged manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15366–15372. IEEE, 2024.

[37] H. Xue, C. Pan, Z. Yi, G. Qu, and G. Shi. Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4974–4981. IEEE, 2025.

[38] K. Zakka, Y. Tassa, and MuJoCo Menagerie Contributors. MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo, 2022. URL http://github.com/google-deepmind/mujoco_menagerie.

[39] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[40] C. He, X. Liu, G. M. S. Camps, J. Bruno, G. A. Sartoretti, and M. Schwager. Demystifying robot diffusion policies: Action memorization and a simple lookup table alternative. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

[42] N. Hansen, Y. Akimoto, and P. Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, Feb. 2019. URL https://doi.org/10.5281/zenodo.2559634.

[43] T. Chen, Z. He, and M. Ciocarlie. Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning. *arXiv preprint arXiv:2008.04460*, 2020.

[44] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal. An end-to-end differentiable framework for contact-aware robot design. *arXiv preprint arXiv:2107.07501*, 2021.

[45] C. Rajani, K. Arndt, D. Blanco-Mulero, K. S. Luck, and V. Kyrki. Co-imitation: learning design and behaviour by imitation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6200–6208, 2023.

[46] N. Liu, S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum. Compositional visual generation with composable diffusion models. In *European conference on computer vision*, pages 423–439. Springer, 2022.

[47] Y. Du, C. Durkan, R. Strudel, J. B. Tenenbaum, S. Dieleman, R. Fergus, J. Sohl-Dickstein, A. Doucet, and W. S. Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. In *International conference on machine learning*, pages 8489–8510. PMLR, 2023.

[48] M. Kodnongbua, I. G. Y. Lou, J. Lipton, and A. Schulz. Computational design of passive grippers. *arXiv preprint arXiv:2306.03174*, 2023.

[49] H. Ha, S. Agrawal, and S. Song. Fit2form: 3d generative model for robot gripper form design. In *Conference on Robot Learning*, pages 176–187. PMLR, 2021.

[50] M. Li, R. Antonova, D. Sadigh, and J. Bohg. Learning tool morphology for contact-rich manipulation tasks with differentiable simulation. *arXiv preprint arXiv:2211.02201*, 2022.

[51] R. Liu, J. Liang, S. Sudhakar, H. Ha, C. Chi, S. Song, and C. Vondrick. Paperbot: Learning to design real-world tools using paper, 2024.

[52] K. R. Allen, T. Lopez-Guevara, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, J. Hamrick, and T. Pfaff. Physical design using differentiable learned simulators. *arXiv preprint arXiv:2202.00728*, 2022.

[53] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina, M. Lepert, Y. J. Ma, P. T. Miller, J. Wu, S. Belkhale, S. Dass, H. Ha, A. Jain, A. Lee, Y. Lee, M. Memmel, S. Park, I. Radosavovic, K. Wang, A. Zhan, K. Black, C. Chi, K. B. Hatch, S. Lin, J. Lu, J. Mercat, A. Rehman, P. R. Sanketi, A. Sharma, C. Simpson, Q. Vuong, H. R. Walke, B. Wulfe, T. Xiao, J. H. Yang, A. Yavary, T. Z. Zhao, C. Agia, R. Baijal, M. G. Castro, D. Chen, Q. Chen, T. Chung, J. Drake, E. P. Foster, J. Gao, V. Guizilini, D. A. Herrera, M. Heo, K. Hsu, J. Hu, M. Z. Irshad, D. Jackson, C. Le, Y. Li, K. Lin, R. Lin, Z. Ma, A. Maddukuri, S. Mirchandani, D. Morton, T. Nguyen, A. O'Neill, R. Scalise, D. Seale, V. Son, S. Tian, E. Tran, A. E. Wang, Y. Wu, A. Xie, J. Yang, P. Yin, Y. Zhang, O. Bastani, G. Berseth, J. Bohg, K. Goldberg, A. Gupta, A. Gupta, D. Jayaraman, J. J. Lim, J. Malik, R. Martín-Martín, S. Ramamoorthy, D. Sadigh, S. Song, J. Wu, M. C. Yip, Y. Zhu, T. Kollar, S. Levine, and C. Finn. Droid: A large-scale in-the-wild robot manipulation dataset. 2024.

[54] OpenAI. Learning to reason with LLMs, Sept. 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

[55] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[56] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[57] B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[58] M. Ballon, A. Algaba, and V. Ginis. The relationship between reasoning and performance in large language models–o3 (mini) thinks harder, not longer. *arXiv preprint arXiv:2502.15631*, 2025.

[59] Y. Wu, Z. Sun, S. Li, S. Welleck, and Y. Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

[60] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[61] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, K. Ghasemipour, C. Finn, and A. Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

[62] H. Ha and S. Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33. PMLR, 2022.

[63] S. Ha, S. Coros, A. Alspach, J. M. Bern, J. Kim, and K. Yamane. Computational design of robotic devices from high-level motion specifications. *IEEE Transactions on Robotics*, 34(5): 1240–1251, 2018.

[64] H. T. Suh, M. Simchowitz, K. Zhang, T. Pang, and R. Tedrake. Pathologies and challenges of using differentiable simulators in policy optimization for contact-rich manipulation. In *ICRA 2022 Workshop: Reinforcement Learning for Contact-Rich Manipulation*, 2022.

[65] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*, 2022.

[66] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane. Computational co-optimization of design parameters and motion trajectories for robotic systems. *The International Journal of Robotics Research*, 37(13-14):1521–1536, 2018.

[67] J. Külz, S. Ha, and M. Althoff. A design co-pilot for task-tailored manipulators. *arXiv preprint arXiv:2509.13077*, 2025.

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

[69] G. Authors. Genesis: A generative and universal physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis.

[70] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[71] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023.

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

[73] G. B. Margolis and P. Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. In *Conference on Robot Learning*, pages 22–31. PMLR, 2023.

[74] K. Sims. Evolving virtual creatures. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 699–706. 2023.

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

[76] K. De Jong. Evolutionary computation: a unified approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 373–388, 2017.

[77] C. G. Langton. Artificial life: An overview. 1997.

[78] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *ACM SIGEVOlution*, 7(1):11–23, 2014.

[79] 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.

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

[81] 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.

[82] J. Hu, J. Whitman, M. Travers, and H. Choset. Modular robot design optimization with generative adversarial networks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4282–4288. IEEE, 2022.

[83] 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.

[84] 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.

[85] K. Fay, D. A. Djapri, A. Zorin, J. Clinton, A. E. Lahib, H. Su, M. T. Tolley, S. Yi, and X. Wang. Cross-embodied co-design for dexterous hands. *arXiv preprint arXiv:2512.03743*, 2025.

[86] Z. Mandi, Y. Weng, D. Bauer, and S. Song. Real2code: Reconstruct articulated objects via code generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[87] Wikipedia. Parallel axis theorem — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Parallel%20axis%20theorem&oldid=1327910993, 2026. [Online; accessed 05-February-2026].

[88] T. E. Truong, M. Piseno, Z. Xie, and K. Liu. Pdp: Physics-based character animation via diffusion policy. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–10, 2024.

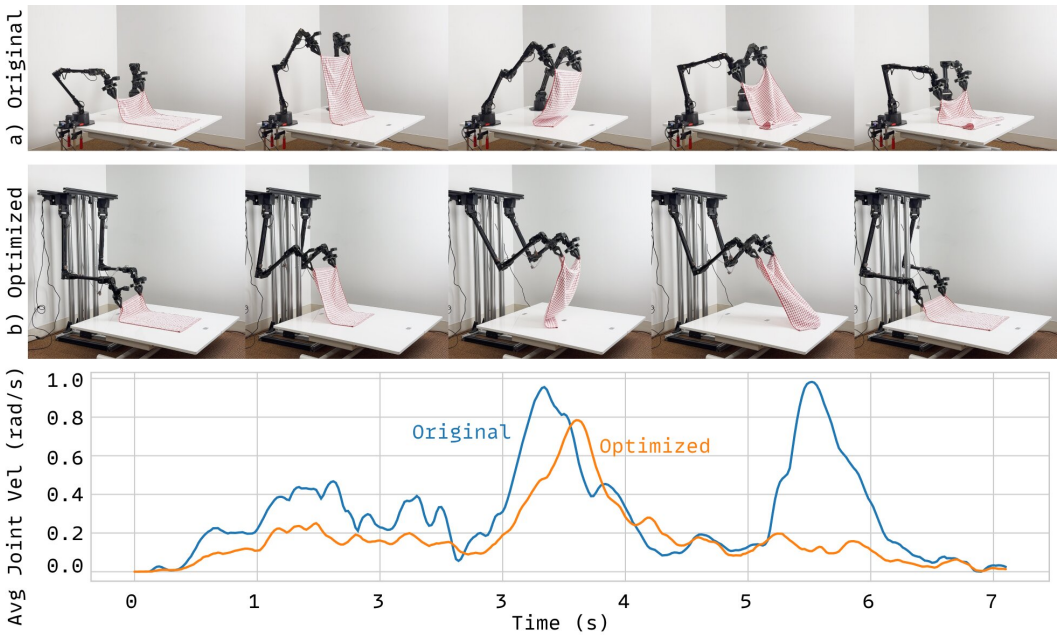

Figure 10: **Real World Unfolding Sequence**. Compared to the original ALOHA design (a), we observe a lower max joint speed and less peaks in our model's optimized design (b).

In accordance with the official CoRL author instructions, we include the appendix with the main paper for the reviewer's convenience.

7 Additional Experiments

In the context of robot co-design, a learned cross-embodiment controller is practically useful only for validating robot embodiments that benefit from costly learned controllers, such as legged robots. However, robots with fast (oracle) kinematic controllers provide faster data generation and thus represent a practical design space to study cross-embodiment control. In this section, we include capacity ablations when training a unified model, with cross-embodiment control (§ 7.1) and motion-to-robot (§ 7.2) experiments.

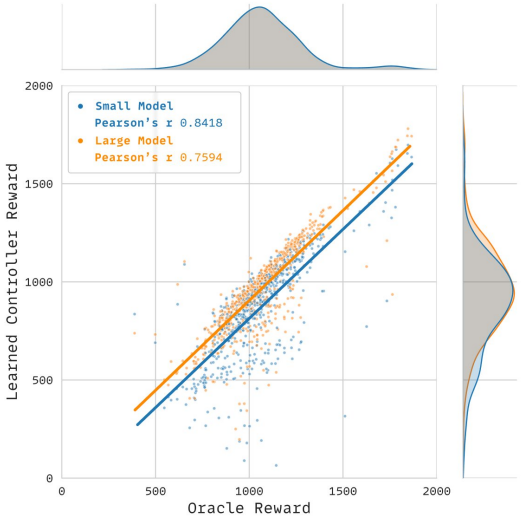

Figure 11: **Bimanual Mobile Control Validation**. For each embodiment, we control it using our cross-embodiment controllers (small/large) and the oracle Mink [27] controller. Our controllers have strong correlation with the oracle's performance. In other words, using our cross-embodiment controllers to evaluate robot designs is faithful to the oracle robot evaluation.

7.1 Bimanual Mobile Cross-Embodiment Control

Setup. We train two model sizes, a small model with 11.6M parameters and a large model with 63.6M parameters, on both the motion-to-robot task and the cross-embodiment controller task. We evaluate the performance for the controller on procedurally generated robots in the mobile bimanual design space using the 26 validation trajectories from the UMI [1] bimanual dish washing dataset. For each trajectory, we include 5 continuous variations per discrete variation of the robot, yielding 25 robots over 650 episodes in total. Since these robots are randomly generated, some embodiments with suboptimal designs exhibit poor control performance for all control methods (hence the higher tracking error).

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival %↑	Reward ↑
Ours (Small)	6.6	10.7	98.9	884.9
Ours (Large)	5.9	9.6	99.5	957.0
Mink (Oracle)	4.8	7.6	100.0	1064.3

Table 1: **Controller Performance against Oracle**

Results. We report the tracking reward defined in equation 1, along with other tracking metrics, in Table 1. In Fig. 11, we observe a strong positive correlation between the tracking performance of our controllers and the oracle Mink controller. We normalize for the number of gradient steps, and report that the larger model is closer to oracle performance in terms of both tracking metrics and Pearson's correlation. However, when normalizing for training FLOPs, the smaller model should receive $5.5\times$ as many training steps. We leave this investigation, along with other model distillation techniques, for future work.

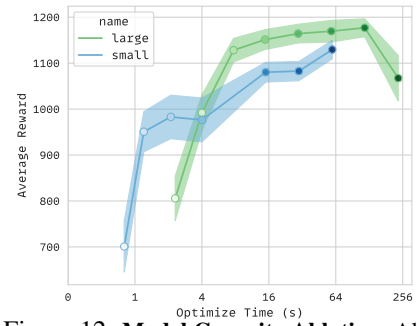

Figure 12: **Model Capacity Ablation**. Although larger models perform better, their inference time also takes longer.

7.2 Bimanual Mobile Motion-to-Robot Optimization

Setup. We use the unified models with 11.6M and 63.6M parameters from the previous section (§ 7.1) and apply the Zeroth Order optimizer for the multi-trajectory optimization setting. As in the main paper, each robot is optimized for all 26 validation trajectories in the UMI dish washing dataset simultaneously, and we repeat for 9 inference seeds. Unlike in the main paper, we use our learned controller (§ 7.1) instead of the oracle controller. We focus these results on the "Tracking Only" reward.

Results. From Fig. 12, we observe that the large model performs better. However, for the same number of parallel samples, the large model requires $4\times$ longer runtime.

8 Additional Metrics from Main Experiments

This section defines all reward terms used for the experiments and reports reward-specific metrics for the experiments presented in Figure 5 in the main paper.

8.1 Tracking Only

Reward Definition. For a given target pose trajectory $\{p^t_{target}, o^t_{target}\}_{t\in T}$ and an achieved robot end-effector pose trajectory $\{p^t_{achieved}, o^t_{achieved}\}_{t\in T}$ each of length T, we compute the position error ϵ^t_p as $||p^t_{target} - p^t_{achieved}||_2$ and the orientation error ϵ^t_o as $\arccos\left(\frac{\text{Tr}(R)-1}{2}\right)$ where R is the rotation from $o^t_{achieved}$ to o^t_{target}. Based on these tracking errors per time step, the tracking reward for an episode of length T is:

$$\sum_t^T \exp\left(-\frac{||\epsilon^t_p||^2}{\sigma_p} - \frac{||\epsilon^t_o||^2}{\sigma_o}\right) \tag{1}$$

Intuitively, the reward is 1 if both position and orientation errors are zero, and less than 1 if either error is non-zero. σ_p and σ_o determine the reward falloff as tracking error increases and are set to 0.01 and 0.5, respectively, for all experiments. Exponentiating the negative tracking error is a common reward function form used in reinforcement learning for robotics [31, 68, 73]. An episode is terminated if the robot falls over (in the case of the quadruped) or if ϵ^t_p exceeds a maximum allowed position error threshold. This threshold is set to 50cm for the ViperX and mobile bimanual design spaces, and 80cm for the quadruped design space. For the bimanual case, we define ϵ^t_p and

ϵ_o^t to be the sum of tracking position and orientation errors over both end-effectors. This means high reward is only possible if position and orientation errors for both end-effectors are low.

Reward-Specific Metrics. We report position and orientation tracking error averaged over all time steps and all end-effectors, the optimization time in seconds, and the survival rate (proportion of episodes that did not terminate early). For each approach, we use the value of n that achieves the highest reward. For CMA-ES, n is the maximum number of rollouts per trajectory. For Zeroth Order and Dynamics Self-Guidance (DGS), n is the number of samples.

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival %↑	Optimize Time *sec*↓
Random	19.4	8.9	72.2	-
CMA-ES	5.0	5.7	98.3	47.5
Zeroth	**4.1**	4.2	96.1	**0.5**
DGS	**4.1**	**3.9**	**99.4**	2.8

Table 2: **ViperX, Single Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival %↑	Optimize Time *sec*↓
Random	17.0	9.4	80.6	-
CMA-ES	7.1	6.1	86.7	663.8
Zeroth	3.4	3.9	95.6	**1.2**
DGS	**2.4**	**3.5**	**98.9**	43.5

Table 3: **ViperX, Multi-Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival %↑	Optimize Time *sec*↓
Random	3.4	8.4	96.1	-
CMA-ES	**1.9**	**5.8**	**100.0**	265.5
Zeroth	2.6	7.4	99.4	0.8
DGS	2.6	7.4	99.4	5.0

Table 4: **Quadruped, Single-Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival %↑	Optimize Time *sec*↓
Random	2.8	4.4	99.6	-
CMA-ES	**1.7**	**2.5**	**100.0**	11505.2
Zeroth	1.8	**2.5**	99.6	**20.7**
DGS	**1.7**	2.7	99.1	30.8

Table 5: **Bimanual, Multi-Trajectory**

8.2 Tracking Torque

Reward Definition. Let τ^t be the total torque in all of the robot's motors at timestep t. The tracking torque reward is:

$$\sum_t^T \max\left(\exp\left(-\frac{||\epsilon_p^t||^2}{\sigma_p} - \frac{||\epsilon_o^t||^2}{\sigma_o}\right) - \alpha_{torque}||\tau^t||^2, 0\right) \tag{2}$$

where the torque weight α_{torque} is set to 5×10^{-5} in all experiments. If the torque is too large, it will outweigh the non-negative tracking term. Avoiding negative rewards is common practice [31, 68, 73], preventing scenarios where large negative rewards encourage agents to terminate episodes early.

Reward-Specific Metrics. In addition to the tracking metrics (§ 8.1), we also report averaged $||\tau^t||_2$.

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Torque *Nm*↓	Optimize Time *sec*↓
Random	19.4	8.9	72.2	**3.8**	-
CMA-ES	5.9	5.7	95.0	4.1	45.4
Zeroth	**4.1**	4.5	97.2	4.4	**0.5**
DGS	**4.1**	**4.1**	**98.3**	4.5	1.9

Table 6: **ViperX, Single Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Torque *Nm*↓	Optimize Time *sec*↓
Random	17.0	9.4	80.6	**4.0**	-
CMA-ES	7.5	5.9	89.4	4.4	644.4
Zeroth	3.4	3.9	95.6	4.6	**1.2**
DGS	**2.5**	**3.5**	**98.9**	4.6	43.4

Table 7: **ViperX, Multi Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Torque *Nm*↓	Optimize Time *sec*↓
Random	3.4	8.4	96.1	4.6	-
CMA-ES	**2.0**	**6.0**	**100.0**	3.2	250.0
Zeroth	2.5	7.1	99.4	**3.1**	**3.3**
DGS	2.5	7.0	98.9	3.2	5.5

Table 8: **Quadruped, Single Trajectory**

8.3 Tracking Velocity

Reward Definition. This reward follows equation 2, replacing $\alpha_{torque}||\tau^t||^2$ with $\alpha_{velocity}||\dot{q}^t||^2$, where $\dot{q}^t$ is each actuator's velocity at time step t. $\alpha_{velocity}$ is set to 0.1 for the ViperX, 0.5 for the mobile bimanual design space, and 0.005 for the quadruped design space.

Reward-Specific Metrics. In addition to the tracking metrics (§ 8.1), we also report averaged $||\dot{q}^t||_2$.

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Velocity *rad/s*↓	Optimize Time *sec*↓
Random	19.4	8.9	72.2	0.391	-
CMA-ES	6.1	5.8	92.5	**0.366**	38.5
Zeroth	4.1	4.3	**98.9**	0.397	**0.5**
DGS	**3.9**	**4.1**	97.2	0.385	1.5

Table 9: **ViperX, Single Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Velocity *rad/s*↓	Optimize Time *sec*↓
Random	17.0	9.4	80.6	0.384	-
CMA-ES	8.5	7.0	87.8	0.385	615.7
Zeroth	**3.4**	**3.7**	**97.8**	**0.362**	**7.3**
DGS	5.5	4.2	**97.8**	0.394	43.4

Table 10: **ViperX, Multi Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Velocity *rad/s*↓	Optimize Time *sec*↓
Random	3.4	8.4	96.1	0.561	-
CMA-ES	**2.1**	**5.9**	**100.0**	**0.376**	251.1
Zeroth	2.8	8.0	98.9	0.450	**1.1**
DGS	3.5	9.0	99.4	0.455	9.1

Table 11: **Quadruped, Single Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Velocity *rad/s*↓	Optimize Time *sec*↓
Random	2.8	4.4	**99.6**	0.252	-
CMA-ES	**1.8**	**2.4**	**99.6**	**0.188**	11505.2
Zeroth	1.9	2.9	99.1	0.271	61.95
DGS	1.9	2.9	98.7	0.247	**53.5**

Table 12: **Bimanual, Multi Trajectory**

8.4 Tracking Size

Reward Definition. To capture the notion of the robot's size $s_{achieved}$, we use the sum of the lengths of all of the robot's geometries, where the length of a geometry is the magnitude of its longest dimension. Since different robot types have very different sizes, we use a target size s_{target} and define the size penalty as:

$$-T\alpha_{size} \min\left(s_{achieved} - s_{target}, 0\right) \tag{3}$$

This term is zero only if the size of the robot is below the target size, and is added to equation 1. We use $\alpha_{size} = 0.1$ for all design spaces, while tuning s_{target} to be 2.0m for the ViperX design space, 6.5m for the quadruped design space, and 5.0m for the mobile bimanual design space.

Guidance Scale. Since diffusion training requires the model to look at its previous diffusion time step's noisy token to predict how to denoise, we find that the attention of a token to itself is significantly higher than its attention to other tokens. This means the gradients of rewards that use embodiment tokens (e.g., size, weight) are also significantly higher than those of rewards that use dynamics tokens (e.g., tracking error). We found that not accounting for this magnitude difference led to invalid robot outputs from diffusion. To address this, we use $\alpha_{size} = 0.005$ for the guidance term, but keep $\alpha_{size} = 0.1$ for the actual reward reported.

Reward-Specific Metrics. In addition to the tracking metrics (§ 8.1), we report the size of the robot as defined above.

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Size *m*↓	Optimize Time *sec*↓
Random	19.4	8.9	72.2	2.35	-
CMA-ES	6.2	6.3	94.4	2.21	38.3
Zeroth	**4.3**	**4.3**	**96.7**	2.31	**0.5**
DGS	4.6	4.4	96.1	**2.19**	1.9

Table 13: **ViperX, Single Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Size *m*↓	Optimize Time *sec*↓
Random	17.0	9.4	80.6	2.42	-
CMA-ES	7.4	6.7	91.1	**2.28**	564.6
Zeroth	3.3	3.8	96.7	2.32	**1.2**
DGS	**2.8**	**3.7**	**98.9**	2.29	3.6

Table 14: **ViperX, Multi Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Size *m*↓	Optimize Time *sec*↓
Random	3.4	8.4	96.1	9.25	-
CMA-ES	**2.3**	**7.2**	**100.0**	7.28	263.3
Zeroth	4.2	11.6	94.4	7.40	**3.2**
DGS	4.6	12.6	94.4	**7.17**	9.0

Table 15: **Quadruped, Single Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Size *m*↓	Optimize Time *sec*↓
Random	2.8	4.4	99.6	4.84	-
CMA-ES	1.8	**2.7**	**100.0**	4.79	11505.2
Zeroth	1.7	**2.7**	99.1	**4.71**	**10.7**
DGS	**1.6**	2.8	99.6	4.73	58.9

Table 16: **Bimanual, Multi Trajectory**

8.5 Tracking Weight

Reward Definition. The robot's total mass $m_{achieved}$ is the sum of all of its geometries' masses. Like equation 3, we set a target mass m_{target}, and scale it by the number of timesteps:

$$-T\alpha_{mass} \min\left(m_{achieved} - m_{target}, 0\right) \tag{4}$$

For the ViperX design space, α_{mass} is set to 10, with a target weight of 3.2kg. This term is added to equation 1.

Guidance Scale. Like in § 8.4, we use $\alpha_{mass} = 0.005$ to account for higher attention of tokens to themselves.

Reward-Specific Metrics. In addition to the tracking-only metrics (§ 8.1), we report the total mass of the robot.

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Weight *kg*↓	Optimize Time *sec*↓
Random	19.4	8.9	72.2	3.2	-
CMA-ES	13.2	6.3	80.6	3.2	41.8
Zeroth	8.7	6.3	91.7	3.0	1.0
DGS	9.2	6.1	92.2	3.0	2.9

Table 17: **ViperX, Single Trajectory**

Approach	Pos Err *cm*↓	Orn Err *deg*↓	Survival *%*↑	Weight *kg*↓	Optimize Time *sec*↓
Random	17.0	9.4	80.6	3.3	-
CMA-ES	17.4	5.6	73.3	3.2	516.4
Zeroth	8.6	**5.1**	**97.2**	**3.1**	**7.3**
DGS	**8.1**	5.7	91.7	**3.1**	45.3

Table 18: **ViperX, Multi Trajectory**

9 Additional Method Details

9.1 RoboToken

Token Count Comparison. The text token count in Figure 4 in the main paper is calculated from the robot's MJCF format tokenized with GPT-4o's tokenizer. We compared against text tokens as these are the only other complete robot representation that can be used directly both as input to and output from a neural network.

Inertia Splitting. To maintain consistency in *RoboToken*, we require that all inertial properties arise strictly from the link's geometry primitives. When a user-provided robot description contains a link with multiple geometries but a single lumped inertia tensor, we apply a heuristic algorithm to split this lumped inertia into individual geometry inertias.

1. We assume uniform density ρ across all primitives within a link.
2. For each primitive i, we calculate its local inertia I_i and mass m_i based on its geometry (e.g., box, sphere, or cylinder).
3. We apply the Parallel Axis Theorem [87] to shift these inertias to the link's common center of mass:
$$I_{\text{total}} = \sum (I_i + m_i[(\mathbf{r}_i \cdot \mathbf{r}_i)\mathbf{E} - \mathbf{r}_i \otimes \mathbf{r}_i])$$
where $\mathbf{r}_i$ is the displacement vector from the primitive center to the link center.

Transform Canonicalization. To ensure that the model does not have to learn redundant spatial offsets, we preprocess all MJCF/URDF files to collapse all transforms into the joint token's transforms. This means all geometries are effectively at the origin of their link's frame.

Attribute Encoding. Robot embodiment attributes can span different data types (boolean, enums, integers, continuous values) and ranges. For all boolean/enum/integer attributes (such as joint types, geometry types, or ID attributes), we use binary encoding. For continuous-valued attributes that range over many orders of magnitude (such as inertia, motor gains), we use log or signed log encoding. We include the RoboToken schema along with each attribute's encoding in Table 19.

Context Length Considerations. Since self-attention is $O(n^2)$ in sequence length, carefully determining the model's context length is critical for training and inference speeds. Therefore, we remove all link pose observations except for two types of links. First are free links in free-base robots (e.g., quadrupeds). Second are end-effectors, which are referred to in the schema as "Track Link" (Table 19) to allow generalization to other robot body parts. Since a robot contains many more links than are usually task-relevant, omitting all these other links significantly reduces the context length

for the dynamics token portion of input tokens. For each design space, we train using that design space's max sequence length.

9.2 Transformer Transformer

Hyperparameters. Our model and training hyperparameters are listed in Tab. 20. We found that when training a unified model, a larger model trained significantly faster. We opted for doubling the hidden dimension and number of heads, and increasing the number of layers to 12 (large model, Fig. 11). To help this larger model train stably, we use a smaller learning rate of 1×10^{-4}.

Joint Token Ordering. We represent each joint with two embodiment tokens, with each part responsible for pointing to one link and encoding the transform to that link. All other attributes of the joint are duplicated between the two tokens. To help the diffusion model distinguish each joint's two tokens, we add a joint ordering ID attribute, which is 1 for the token with the higher link ID. Joint order IDs are converted into a learnable positional embedding and added to the joint tokens.

Dynamics Model Temporal Resolution. We experimented with including more episode timesteps in the model context, hypothesizing that increased temporal resolution would allow the model to learn a more accurate dynamics model. However, these models did not significantly improve downstream hardware optimization performance, yet they significantly increased training and inference time. Thus, we opted for 8 time steps, which we empirically observed to strike the best balance between speed and performance for our experiments.

Diffusion Process Length. We also found that while modeling dynamics benefits from longer diffusion processes, modeling actions with shorter diffusion processes was sufficient. Specifically, precise state predictions for downstream reward-guided diffusion, especially for the tracking error objective, are critical for correct guidance, and a longer diffusion process achieves lower prediction error. Meanwhile, action prediction is robust to small deviations from the expert. Thus, we use 100 and 5 time steps for motion-to-robot and cross-embodiment control, respectively.

Explicit Padding Tokens. For motion-to-robot, the model can generate sequences with variable lengths depending on the number of links, joints, and actuators of the robot it will generate, so the model is trained (without masking in the attention mechanism) to explicitly predict padding tokens, which are special-valued tokens detectable by our detokenizer. Meanwhile, for cross-embodiment control, all padding tokens are masked out.

Checkpoint Selection. We use the best checkpoints, which are typically among the last few checkpoints. We found that although motion-to-robot optimization performance improves quickly early in training, cross-embodiment control performance plateaus only in later checkpoints.

9.3 Data

9.3.1 Procedural Generation

We define a procedural generation grammar that grows robots from a common root component outwards. We extend the MuJoCo MJCF XML format to support a meta-level language in the custom data XML fields and inject a parser for handling such meta-level fields. This additional meta-level functionality allows users to define randomized continuous attributes for a subcomponent (e.g., dimension of a leg), constraints between attributes (e.g., length of left calf should equal length of right calf), as well as randomized discrete attributes (e.g., rotation axis of a joint, inclusion of a spring-loaded leg linkage mechanism). Thus, given a procedural robot design space, each robot embodiment can be parameterized by a list of discrete choices and a list of continuous choices, where the length of both lists is variable (some discrete choices can alter the degrees of freedom of the robot). Using our system, existing robot definitions can be easily extended to support a wide range of procedural geometry, kinematics, and dynamics variation, generating the source of embodiment diversity in our dataset.

ViperX. For discrete variations, we include 4 different mounting orientations (left, right, upright, downward), 3 different DoF variations (5 DoF, 6 DoF, and 7 DoF), and 3 joint rotation axis variations per upper joint (X-axis, Y-axis, and Z-axis rotation). For continuous variations, we include mounting positions (within 0.7m, 1.0m, and 0.6m for XYZ directions respectively), mounting Z-angle ($-\pi/2$ rad to $+\pi/2$ rad), and link length variations (-5 cm to $+20$ cm relative increase).

Quadruped Manipulator. For discrete variations, we include 2 arm mount options (fixed or sliding linear rail along the back), 2 leg design options per leg (spring-loaded linkage or serial leg, chosen independently for all 4 legs), and knee direction (flipped forward vs. flipped backward, shared within the front pair and within the back pair), yielding 7 binary design choices in total. For continuous variations, we include arm link length variations (-10 cm to 40 cm relative to initial length), arm mounting position (-20 cm to $+20$ cm), body length (10 cm to 50 cm), leg link lengths (20 cm to 60 cm), leg mounting Z-angle per leg (-0.7 rad to 0.7 rad), battery placement along the length of the robot (heavy green box at the core of the robot, -20 cm to $+20$ cm).

Mobile Bimanual. For discrete variations, we include three spine design choices (a fixed spine, a telescopically sliding spine, and a bending spine) and a torso 45° pitching DoF (included or not, allowing the robot to lean forward). For continuous variations, we include spine length variations (60 cm to 140 cm for the fixed spine, 10 cm to 50 cm for the telescopically sliding spine, and 20 cm to 80 cm for the bending spine), shoulder offset (4 cm to 10 cm in the y direction, -3 cm to 8 cm in the z direction), and arm link lengths (2 cm to 8 cm for the shoulder link, 1 cm to 18 cm for the arm and forearm).

ALOHA. For discrete variations, we include three mounting point orientations (upright facing each other, vertically mounted on either side, or upside down). For continuous variations, we include mounting position variation (-40 cm to 0 cm for x, 40 cm to 60 cm for inter-arm distance, and 60 cm to 90 cm for the z height), as well as arm length variations (10 cm to 50 cm for the arm extension, 5 cm to 25 cm for the forearm extension). We constrain the bimanual set up to be symmetric about the XZ plane.

9.3.2 Whole-Body Controller (WBC) Training

Following the WBC controller formulation from UMI on Legs [31], we train our WBC using PPO [30]. This follows a standard RL controller training procedure that uses a task reward (tracking in our case, equation 1) with regularization penalty terms (joint velocity, acceleration, limits, body orientation, etc.) and in-training perturbations (random kicks, transports) to make the policy robust. In addition to the robot's states, these RL experts also observe the continuous variation parameters of the robot's embodiment (§ 9.3.1) and 4 target poses into the future. At the beginning of each episode, a random trajectory from the training set of end-effector trajectories is selected, a random transform augmentation is applied, and the robot is then controlled to track the target trajectories. Episodes are terminated if the robot falls over, with a termination penalty applied. We train one RL expert per discrete choice, which for the 7 binary design choices in the quadruped case yields 128 specialized experts. Training each RL policy takes 16 hours on an NVIDIA A100. At data generation time, we procedurally generate robots from a stream of randomized discrete and continuous choices, load the corresponding RL policy, and control the robot to track a training trajectory. Each rollout is then tokenized into a RoboTokens dataset. We add control perturbations to the expert actions to increase state diversity, but log the unperturbed expert action for supervision [88].

9.3.3 Data Quality

We ensure high tracking performance in training datasets in two ways. First, we filter all episodes that last less than 100 time steps (i.e., 2 seconds), which represents embodiments that fell over (for quadrupeds) or exceeded the tracking position error termination threshold during that time. Second, we bias the data generation towards higher-performance embodiments for the ViperX and ALOHA design space. While tracking error is reasonable for the quadruped and mobile bimanual design spaces, the tracking error of the random ViperX and ALOHA embodiment is very high (19.4 cm, 8.9°, Table 2) since the kinematics of fixed-base arms critically determine whether a target pose is even reachable. To improve training embodiment quality, we run CMA-ES on the training trajectories with a population of 5 for 3 generations and use all embodiments sampled during CMA-ES as training data. Although improving training data quality is one way CMA-ES and our approach can be paired together, we believe their integration could also occur at inference time, where our algorithm acts as the sampler for an evolutionary algorithm. We leave this investigation for future work.

9.3.4 Dataset Size

Our ViperX dataset includes 3.8M episodes with 2B time steps in total. Our quadruped dataset includes 1.3M episodes with 500M time steps in total. Finally, our bimanual mobile dataset includes 50K episodes and 69M time steps in total. In each dataset, each robot embodiment design is used for 10 episodes, so the total number of training embodiments is 380K, 130K, and 5K for the ViperX, quadruped, and bimanual design spaces respectively.

9.4 Inference

Stochastic DDIM sampling. DDIM [22] has a hyperparameter for controlling the amount of noise injected into the sampling procedure, typically denoted as η in DDIM implementations. Although deterministic DDIM sampling ($\eta = 0.0$) is common, we found that using stochastic sampling with $\eta = 1.0$ for Dynamics Self-Guidance (DGS) specifically improved motion-to-robot optimization performance significantly. Meanwhile, tuning η did not affect our Zeroth Order sampler's performance. This suggests that, like all first-order methods, our DGS sampler is prone to local minima, but injecting a gradually decaying noise term aids exploration along the diffusion process.

Guidance Scale. For DGS, we use the highest guidance scale that does not push the diffusion model out of distribution to generate invalid robots. It is set to 50, 100, and 0.2 for the ViperX, quadruped, and bimanual design spaces respectively.

RoboToken Type	Attribute	Dim	Encoding
Link	Geometry Type	3	Binary
	Geometry Size	3	None
	Mass	1	Log
	Inertia Position	3	None
	Inertia Rotation	9	None
	Diagonal Inertia	3	Log
	Friction	3	None
	Contact Dim.	3	Binary
	Color	4	None
	Free Link ID	1	Binary
	Track Link ID	1	Binary
	Link ID	-	Pos Emb
Dynamic Joint	Joint Type	2	Binary
	Range	2	None
	Armature	1	Log
	Damping	1	Log
	Friction Loss	1	Log
	Stiffness	1	Log
	Spring Reference	1	None
	Position Reference	1	None
	Position to Link	3	None
	Rotation to Link	9	None
	Link ID	7	Binary
	Dyna Joint ID	-	Pos Emb
Fixed Joint	Position	3	None
	Rotation	9	None
	Link ID	7	Binary
	Fixed Joint ID	-	Pos Emb
Actuator	Control Range	2	None
	Force Range	4	Signed Log
	Position Gain	1	Log
	Velocity Gain	1	Log
	Gear Ratio	1	None
	Actuator Type	1	Binary
	Dyna Joint ID	6	Binary
	Actuator ID	-	Pos Emb
Free/Track Link Obs	Position	3	None
	Rotation	9	None
	Free/Track Link ID	-	Pos Emb
	Time ID	-	Pos Emb
Dynamic Joint Obs	Joint Pos	1	None
	Joint Vel	1	None
	Dyna Joint ID	-	Pos Emb
	Time ID	-	Pos Emb
Actuator Obs	Velocity	1	None
	Force	1	None
	Actuator ID	-	Pos Emb
	Time ID	-	Pos Emb
Control	Target Pos	1	None
	Actuator ID	-	Pos Emb
	Time ID	-	Pos Emb
Target Pose	Position	3	None
	Rotation	9	None
	Track Link ID	-	Pos Emb
	Time ID	-	Pos Emb

Table 19: **RoboToken Schema.** Not listed here are ball joints, whose schema is identical to that of dynamic joints', but whose observations are quaternions and angular velocities instead.

Hyperparameter	Value
Hidden dim	256
Num layers	8
Num heads	4
Dropout	0.0
LR	5×10^{-3}
LR ramp up	500 steps
Batch size	64
Weight decay	0.0
EMA power	0.75
Num epochs	50
Num steps per epoch	16,384

Table 20: **Model Hyperparameters**