CURRICULUM-BASED CO-DESIGN OF MORPHOLOGY AND CONTROL OF VOXEL-BASED SOFT ROBOTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Co-design of morphology and control of a Voxel-based Soft Robot (VSR) for solving a given task is challenging due to the bi-level optimization in the enormous combined design and policy space. In this paper, we present a Curriculum-based Co-design (CuCo) method for learning to design and control VSRs through an easy-to-difficult process. Specifically, we expand the design space from a small size to the target size gradually through a predefined curriculum. At each stage of the curriculum, we use reinforcement learning to simultaneously train the design and policy, which is enabled by incorporating the design process into the environment and using differentiable policy representations. The converged morphology, the learned design and control policies from the last stage are inherited and serve as the starting point for the next stage. In the empirical studies, we show that CuCo is more efficient in creating larger robots with better performance by reusing the practical design and control patterns learned within each stage, in comparison to prior approaches that learn from scratch in the space of target size (see video).

1 INTRODUCTION

The philosophy of embodied cognition Pfeifer & Bongard (2006); Pfeifer et al. (2014) inspires the domain of robotics that a robot's ability to interact with the environment depends both on its brain (control policy) and body (morphology), which are inherently coupled Spielberg et al. (2019); Gupta et al. (2021). However, finding an optimal robot morphology and its controller for solving a given task is often unfeasible. The major challenge for this endeavor is the enormous combined design and policy space. Firstly, the freedom to pick the number of multi-material modules and the ways they are connected makes it notoriously difficult to explore the design space Medvet et al. (2022). For instance, in a robot simulator Liu et al. (2020), there are over 4×10^8 possible morphologies for a robot composed of only 12 modules. Secondly, the evaluation of a morphology requires a separate training procedure for its unique controller.

In this work, we consider the co-optimization of design and control of Voxel-based Soft Robots (VSRs) Medvet et al. (2021), a form of modular soft robots composed of elastic, multi-material cubic blocks. Unlike fragile fully-integrated robots, they can be easily disassembled and reassembled to adapt to a wide range of environments Shah et al. (2020); Pigozzi et al. (2022). For efficiently exploring the modular robot design space, prior approaches commonly rely on artificial evolution Sims (1994); Cheney et al. (2013); Bhatia et al. (2021). They maintain a population of design prototypes and adopt a bi-level optimization loop, where the outer loop of morphology optimization is based on the fitness of individual controllers from the inner loop. These methods, however, tend to learn from scratch in the target design space where there is a significant combinatorial explosion. Thus, they spend a large amount of time on policy optimization and evaluation. Additionally, the separate training procedures significantly hinder the experience of design and control to be shared across different robots.

In view of these challenges, we propose a Curriculum-based Co-design (CuCo) method for learning to optimize the morphologies and behaviour of VSRs (Figure 1). CuCo draws inspiration from Curriculum Learning (CL) that starting with easier tasks can help the agent learn better when presented with more difficult tasks later on Bengio et al. (2009). The key to our approach is expanding the design space from a small size to the target size gradually through a predefined curriculum. Precisely, at each stage of the curriculum, we learn practical design and control patterns via Reinforcement

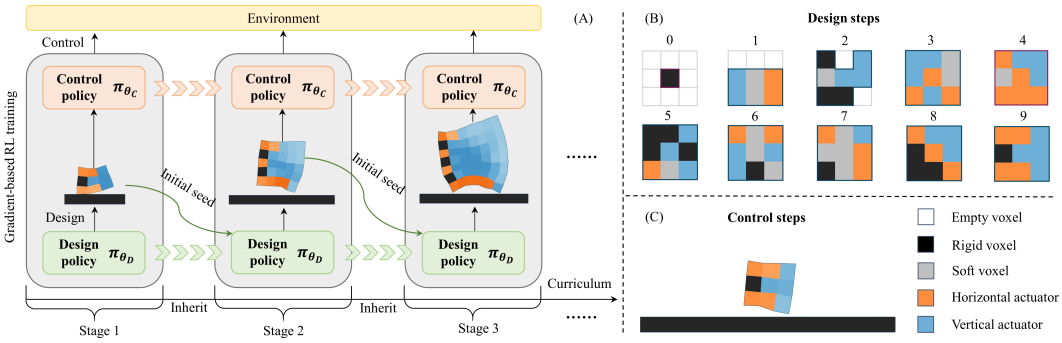

Figure 1: Schematic of our approach CuCo. **(A)** We investigate the brain-body co-design problem of VSRs by establishing an easy-to-difficult optimization process between a small design space (e.g., 3×3) to the target space (e.g., 7×7). **(B)** In 3×3 design space, the design policy performs for a finite number of steps to develop a VSR that is initiated by a single voxel. **(C)** A VSR is composed of five kinds of voxels and actuated by its actuator voxels.

Learning (RL) Schulman et al. (2017), which is enabled by incorporating the design process into the environment and using differentiable policy representations. The converged morphology, the learned design and control policies from the last stage are inherited and serve as the starting point for the next stage. Due to the exploitation of the previous knowledge of design and control, CuCo can quickly produce robust morphologies with better performance in larger dimensional settings.

To handle incompatible state-action spaces and make the control patterns transferable across various morphologies, we model the local observations of all voxels as a sequence and adopt the self-attention mechanism Vaswani et al. (2017) in our control policy to capture the internal dependencies between voxels, which caters to the need for dynamically accommodating changes in morphologies. We also add material information to each voxel's observation, thus the control policy is conditioned on the robot's morphology, which eliminates the need for a population of prototypes and enables experience to be shared across different robots, bringing better sample efficiency and flexibility. We propose to utilize an indirect-encoding Neural Cellular Automata (NCA) Mordvintsev et al. (2020) as the design policy, which takes multiple actions to grow a robot from an initial seed (morphology). NCA encodes complex patterns in a neural network and generates different developmental outcomes while using a smaller set of trainable parameters.

In this paper, we make the following contributions: (1) We propose CuCo, a curriculum-based co-design approach for learning to design and control VSRs. (2) We demonstrate that the practical design and control patterns learned in small design spaces can be exploited to produce high-performing morphologies in larger dimensional settings. (3) We utilize a design policy based on NCA and a self-attention-based control policy to share design and control patterns across different sizes of morphologies, bringing better sample efficiency and flexibility. (4) Comprehensive experimental studies show that CuCo is more efficient in creating larger robots with better performance, in comparison to prior approaches that learn from scratch in the target design space.

2 Related Work

Brain-body Co-design. Finding the optimal morphology and controller for a robot to solve a given task is a challenging subject Sims (1994); Wang et al. (2019b); Nygaard et al. (2020); Medvet et al. (2021); Hejna et al. (2021). Current works can be divided into two categories, gradient-free and gradient-based. Most gradient-free methods rely on artificial evolution to explore the vast design space. Cheney et al. (2013) and Cheney et al. (2018) evolve VSRs with 3D voxels over materials and actuators. Bhatia et al. (2021) create a large-scale benchmark and several population-based co-design algorithms. Walker & Hauser (2021) evolve a sculpting adaptation system that allows VSRs to change their bodies to adapt to the external environment. Talamini et al. (2021) propose a task-agnostic approach for automatically designing morphologies. While the population-based approaches are not easily trapped in local optima, they are more sample inefficient and computa-

tionally expensive. Besides, most of these works rely on simplified open-loop controllers, preventing robots from learning more complex behaviour. In contrast, gradient-based methods, specifically, reinforcement learning, have been conducted for optimizing rigid-bodied robots. Ha (2019) uses a population-based policy gradient method to improve robot design. Pathak et al. (2019) construct a self-organized modular robotic system that can generalize to unseen morphologies and tasks. Luck et al. (2019) learn a critic-based surrogate model for evaluating morphologies. Schaff et al. (2019) maintain a distribution over morphological parameters. Chen et al. (2020) learn a hardware-conditioned policy. Recently, Yuan et al. (2022) represent the design process as a part of the environment and propose a dual policy for design and control. Our work extends this idea to the modular robot design space but differs from all the above-mentioned works where the optimization is directly conducted in the target design space, CuCo starts with a small design space and gradually extends the optimization process to higher dimensional settings, bringing better plasticity and evolvability Hassanien et al. (2009).

Curriculum Learning. Curriculum learning follows the spirit that training the agent with a series of tasks organized according to different complexity can speed up the convergence of the learning process Bengio et al. (2009). It has achieved tremendous success in robotic learning tasks Li et al. (2020); Iscen et al. (2021). One of the keys to curriculum learning is task generation. Wang et al. (2019a) and Wang et al. (2020) generate an endless progression of increasingly challenging environments to evolve robust robots. Fang et al. (2021) propose a method to learn skills via automated generation of diverse tasks. Stensby et al. (2021) co-design robots' morphologies and controllers together with environments evolved open-endedly. We adopt a fixed curriculum by increasing the size of the robot design space according to the learning progress of our method, as the number of possible robot morphologies increases exponentially with the expansion of each dimension of the design space.

Learning a general Controller. Another related line is learning control policies which can generalize across different morphologies. Many such approaches utilize Graph Neural Networks (GNNs) Sanchez-Gonzalez et al. (2018); Wang et al. (2019b); Huang et al. (2020); Yuan et al. (2022) that condition the policy network input on a graph representation related to the robot's morphology. However, for modular robot systems GNNs have the limitation of aggregating multi-hop information Kurin et al. (2021); Pigozzi et al. (2022), which makes the modules not completely interchangeable and the system could be influenced by information loss, greatly limiting their flexibility. Instead, recent approaches using Transformers Vaswani et al. (2017) as universal controllers have shown that generalization can be further enhanced by modularization, due to the success in modeling dynamic structures via the self-attention mechanism Gupta et al. (2022); Trabucco et al. (2022).

3 Background

Voxel-based Soft Robots. We investigate VSRs with multi-material cubic blocks (e.g., soft voxels, rigid voxels and actuator voxels) which are organized in a 2-D grid (Figure 1 (C)). The connectivity describes the connection pairs of adjacent voxels. During the simulation, voxels (except empty voxels) only sense locally, and based on the input sensory information, a controller outputs control signals to vary the volume of actuator voxels. Similar to biological tissues, they can be easily expanded and contracted to complete a wide variety of tasks Bhatia et al. (2021).

Reinforcement Learning. Unlike prior co-design works of VSRs that commonly rely on simplified open-loop controllers, we use Reinforcement Learning (RL) to simultaneously train the design and control policies. In the domain of RL, the problem is usually modeled as a Markov Decision Process (MDP), which can be specified by a 5-tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$. With the state space $\mathcal{S}$ and the action space $\mathcal{A}$, $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the transition function of the environment, $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and the discount factor $\gamma \in (0, 1]$ specifies the degree to which rewards are discounted over time. The goal of the RL agent is to find a policy to maximize the total expected reward $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$, here the policy is represented by a deep neural network parameterized as $\pi_\theta(a_t|s_t)$, which maps from states to distributions over actions. In this work, we employ Proximal Policy Optimization (PPO) Schulman et al. (2017), a popular RL algorithm base on Policy Gradient (PG). PPO uses Kullback-Leibler (KL) divergence to penalize the change between the old and the current policy to avoid instabilities from drastic changes in the policy's behaviour.

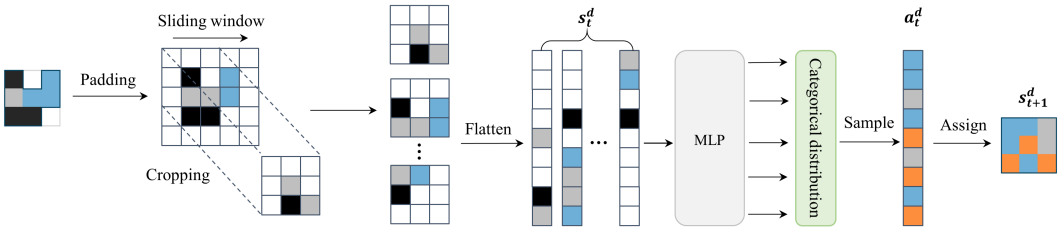

Figure 2: Visualization of a single design step. A multilayer perceptron is incorporated in NCA that determines a cell's state by taking its current state and its neighboring cells' states as the input and outputs a categorical action distribution. By doing so, each cell of the body is updated to the state with the sampled action.

Neural Cellular Automata. Since we incorporate the design process into the environment, our objective is to find a design policy that can deal with a variety of input sizes across different VSRs' morphologies and output a series of actions to modify them. Typically, VSRs can be characterized as multi-cellular systems which develop from a small set of initial cells (voxels) Horibe et al. (2021). We represent our design policy as a Neural Cellular Automata (NCA) Mordvintsev et al. (2020); Sudhakaran et al. (2022); Palm et al. (2022), which begin with some initial seeds and update their states based on local interactions according to rules parameterized by a multilayer perceptron, making it possible for differentiable training guided by objective functions.

Self-attention. Self-attention was proposed due to the challenges in handling long sequences Vaswani et al. (2017). The goal is to dynamically track the relationships between components in the input sequence, and self-attention can be viewed as an adaptive weights mechanism. Given the input vectors X, W^Q, W^K and W^V are three trainable weight matrices. Q, K and V are the query, key, and value vectors where $Q = XW^Q, K = XW^K$ and $V = XW^V$. The importance scores are firstly computed by the dot product between Q and K and then scaled by a non-linear function. Self-attention is typically integrated into the transformer block, which also contains a feed-forward neural network, each of these two submodules has a residual connection around it and is succeeded by a normalization layer. Our work here specifically uses the transformer encoder, enabling our control policy to dynamically adapt to arbitrary dimensional observations due to changes in morphologies.

4 CuCo: Learning to Design and Control from Easy to Difficult

Given a target design space (e.g., 7×7), exploring all possible combinations and training a separate controller to evaluate each design is unfeasible. The key to our approach is to expand this design space from a small size to the target size gradually through a predefined curriculum, which divides the whole optimization process into several stages (Figure 1 (A)). We maintain a design policy π_{θ_D} and a control policy π_{θ_C} from the beginning to the end, thus the overall policy is represented as $\pi_\theta = \{\pi_{\theta_D}(a_t^d|s_t^d, \Psi_t), \pi_{\theta_C}(a_t^c|s_t^c, \Psi_t)\}$ where $\Psi_t \in \{0, 1\}$ is a binary value indicating whether the robot is being created or being controlled, s_t^d and s_t^c represent the state information received at each time step of design and control, respectively. Within each stage, we simultaneously learn to design and control through reinforcement learning, which is enabled by incorporating the design process into the environment. Specifically, at the beginning of each episode, the design policy $\pi_{\theta_D}(a_t^d|s_t^d, \Psi_t)$ performs N_D steps to develop a robot from an initial seed, no reward is assigned (Figure 1 (B)) during the design steps. The resulting robot is then controlled by $\pi_{\theta_C}(a_t^c|s_t^c, \Psi_t)$, finally the two policies are jointly updated using PPO Schulman et al. (2017) according to the future rewards collected in the environment. From one stage to the next, the dimensions of the design space are expanded according to the curriculum and the converged morphology from the last stage will serve as the new initial seed. The learned design and control policies are inherited and later fine-tuned to produce larger and robust bodies. In the remaining section, we describe details about each component of CuCo.

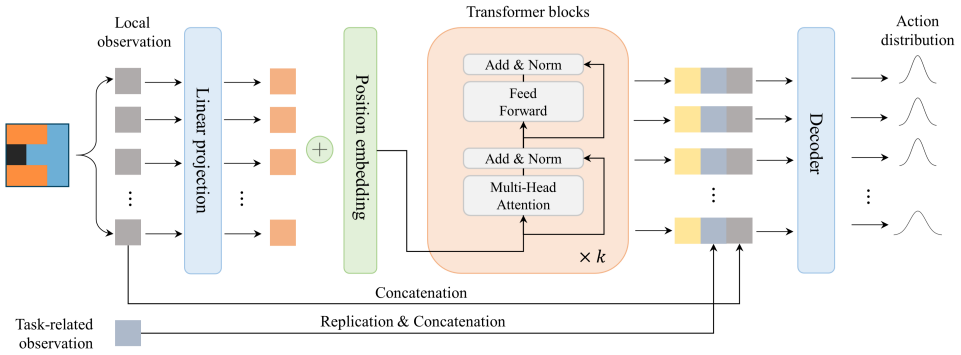

Figure 3: Visualization of the network architecture of the control policy. We model the local observations of all voxels as a sequence, and utilize a Transformer encoder to process the current state.

4.1 Design Policy

Figure 2 demonstrates a single design step when developing a VSR. The design space is surrounded by empty voxels and each voxel is represented as a discrete value that corresponds to its material property (e.g., empty voxel=0, soft voxel=1, rigid voxel=2, horizontal actuator=3 and vertical actuator=4). Thus, the state vector s^{d_i} for each voxel is composed of its type and the types of its neighborhood (e.g., Neumann and Moore neighborhood) Horibe et al. (2021). Practically, let N be the size of the design space, the MDP state of a design step is represented by $s_t^d = \{s_t^{d_1}, s_t^{d_2}, ..., s_t^{d_N}\}$. Here, the design policy $\pi_{\theta_D}(a_t^d|s_t^d, \Psi_t)$ is a multilayer perceptron parameterized by θ_D. It determines a cell's state by taking its current state and its neighboring cells' states as the input and outputs a categorical action distribution. The dimension of the input layer corresponds to the number of the cell's neighbors (e.g, 9 in Figure 2) and the dimension of the output layer corresponds to the total number of material types (e.g, 5 in Figure 2).

At the beginning of each episode of the first stage, a single voxel is used as the initial seed and is placed at the center of the design space, the rest voxels' states are set to zero. After performing the finite design steps described above, all voxels are updated according to the policy's outputs. Note that, in the next learning stage, the converged morphology from the last stage will serve as the new initial seed, which is also placed at the center of the design space. Since the morphology also defines the connectivity to guarantee that the robot does not have any isolated modules. Instead of manually removing these modules that are not connected or connected diagonally, here, we let the design policy learn the connectivity on its own. We don't assign any reward during the design steps and terminate the episode if the resulting design in each step does not meet the connectivity or has no actuator voxels.

4.2 Control Policy

Once a robot is designed, it is consumed by the control policy $\pi_{\theta_C}(a_t^c|s_t^c, \Psi_t)$ to interact with the environment. Our policy is invariant to the dimensionality of observations and actions as we model the local observations of all voxels as a sequence. The input state of the robot at time step t is represented as $s_t^c = \{s_t^v, s_t^g\}$, where $s_t^v = \{s_t^{v_1}, s_t^{v_2}, ..., s_t^{v_N}\}$, $s_t^{v_i}$ is composed of each voxel's local information which contains the relative position of its four corners with respect to the center of mass of the robot and its material information. s_t^g is the task-related observation such as terrain information of the environment and goal-relevant information. The morphology of the robot is unchangeable during the interaction with the environment.

To handle the incompatible state-action spaces and dynamically accommodate changes in morphologies, we incorporate the self-attention mechanism into the control policy, which can control all the agents with a single deep neural network, bringing better sample efficiency than population-based methods. Figure 3 illustrates the architecture of the control policy. We apply a learnable position embedding layer after the linear projection layer. The local observation and the output of the Transformer are concatenated with a task-related observation before passing them through a decoder to

output the corresponding actions. The action distribution is modeled as a Gaussian distribution and a fixed diagonal covariance matrix, and all actions are mapped to the continuous value within the range of $[0.6, 1.6]$.

4.3 POLICY UPDATE AND THE CURRICULUM SETTING

We simultaneously train the design and control policies using Proximal Policy Optimization (PPO), which is based on the Actor-Critic architecture Schulman et al. (2017). In our work, the critic network has the same architecture as the control policy network. It estimates the value function $\mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r]$, which suggests a possible policy distribution. Here, we use the average of the outputs of total voxels as the final predicted value. With the policy gradient method, both design and control policies are updated in the direction towards maximizing the predicted value. When estimating the value of the MDP state of a design step, we set the environment state s^c to 0. We apply a predefined curriculum that adjusts the dimensions of the design space according to the training progress of the agent. We define the hard case according to the dimensions of design space as the number of possible robot designs increases exponentially with each additional block. During training, we increase the dimension when the target number of policy iterations is reached.

5 EMPIRICAL EVALUATION

In this section, we demonstrate the effectiveness of CuCo on various tasks. We seek to answer the following questions: (1) Does CuCo provide an effective mechanism for learning design and control of VSRs? (2) How does CuCo compare to prior approaches which learn from scratch in the space of target size in terms of the final performance and the produced morphologies? (3) What impact does each component of CuCo bring to the learning process?

5.1 ENVIRONMENTAL SETTINGS

We create a modular robot design space as described in Section 4 and several task-related environments[1] based on the Evolution Gym platform, which is a recently proposed benchmark for evolving the structure and control of soft robots Bhatia et al. (2021). We evaluate the performance of CuCo on the following tasks. **Walker**: The target design space is 7×7 and the robot is rewarded by walking as far as possible on flat terrain. **Pusher**: The target design space is 7×7 and the robot is rewarded by pushing a box initialized in front of it as far as possible. **UpStepper**: The target design space is 5×5 and the robot is rewarded by climbing up a stair with different lengths of steps. **Jumper**: The target design space is 7×7 and the robot is rewarded by jumping as high as possible. Note that in all environments, the goal of the robot is to maximize the cumulative reward per episode. Additional details about the environments can be found in Appendix A.

5.2 BASELINES AND ALGORITHM SETTINGS

We compare CuCo with a number of approaches, both RL-based and Evolution-based. The baselines are introduced as follows:

CuCo-NCU: CuCo-NCU is a variant of our proposed approach without the curriculum component, which is similar to the Transform2Act algorithm proposed in Yuan et al. (2022). CuCo-NCU learns to design and control from scratch in the space of target size. This helps us to investigate the effectiveness of the curriculum.

Additionally, we compare CuCo with three evolutionary methods proposed in Bhatia et al. (2021). The inner loop optimization of control policies is driven by PPO. Based on the fitness values of individuals, the outer loop of morphology optimization is implemented by three kinds of state-of-the-art design optimization algorithms.

GA: Genetic Algorithm (GA) directly encodes the VSR's morphology as a vector where each element is tailored to the voxel's material property in order. It keeps a proportion of robots with high

[1]Code will be available online

Table 1: Mean and standard error of all results across environments and baselines. The result with the highest mean is shown in bold. $*$ indicates that the method reaches the upper bound of the task score.

Environment	CuCo	CuCo-NCU	GA	CPPN-NEAT	BO
Walker	$*10.46 \pm 0.01$	8.90 ± 2.01	$*10.45 \pm 0.02$	$*10.46 \pm 0.01$	9.38 ± 1.13
Pusher	$\mathbf{11.74 \pm 1.43}$	1.73 ± 2.19	7.88 ± 1.30	9.66 ± 1.24	8.73 ± 1.16
Jumper	$\mathbf{13.08 \pm 1.09}$	0.79 ± 1.98	1.85 ± 1.31	3.82 ± 2.33	1.28 ± 0.78
UpStepper	$\mathbf{4.40 \pm 0.28}$	2.45 ± 1.70	2.58 ± 0.74	3.65 ± 0.50	2.43 ± 0.53

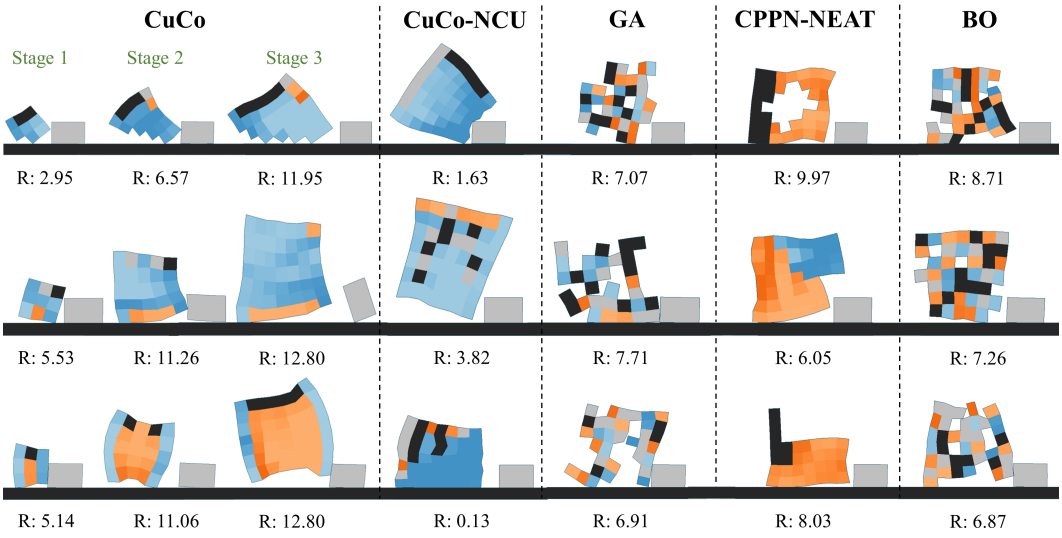

Figure 4: Visualization of converged morphologies of all methods in the Pusher environment across three different runs. Below each VSR is its average performance over five tests. Depictions in other environments can be found in Appendix B.

fitness as survivors at the end of each generation, and the survivors undergo mutation conducted by randomly changing each voxel of the robot with a probability to produce a new generation.

CPPN-NEAT: In this method, the morphology of VSR is indirectly encoded by a Compositional Pattern Producing Network (CPPN) Stanley (2007); Cheney et al. (2013), which relies on having access to the whole design space. The training of CPPN is accomplished by NeuroEvolution of Augmenting Topologies (NEAT) Stanley & Miikkulainen (2002).

BO: Bayesian Optimization (BO) is widely used in solving black-box problems. It trains a surrogate model based on the Gaussian Process (GP) to alleviate the computational burden of expensive fitness evaluation Kandasamy et al. (2018).

Following the convention in the literature Cheney et al. (2013); Horibe et al. (2021), we consider a 2-D design space where all dimensions are equal in size. The curriculum defines three stages ($3 \times 3 \rightarrow 5 \times 5 \rightarrow 7 \times 7$) for Walker, Pusher and Jumper. In UpStepper, the curriculum is set to ($3 \times 3 \rightarrow 5 \times 5$). We run all experiments for each task with the same number of time steps in the environment. For CuCo, this amounts to 1000 policy updates in each stage. We employ a population with a size of 12 for all evolution-based methods and report the performance of the best individual from the population for each task. Further implementation details can be found in Appendix C.

5.3 Overall Performance

In Table 1, we list the final performance of each method and the result is reported over 7 different runs per environment. We can clearly see that CuCo significantly outperforms the baselines, achiev-

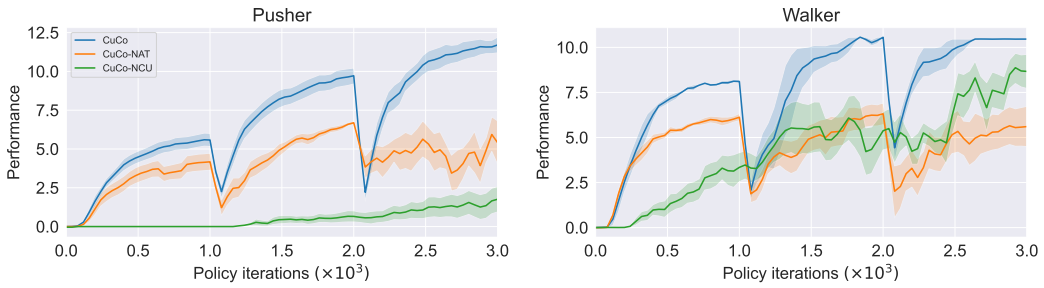

Figure 5: Ablations of the curriculum and the self-attention mechanism. The results suggests that the curriculum an self-attention both contribute to the performance greatly.

ing the highest score in Pusher, Jumper, and UpStepper. In Walker, CuCo, GA, and CPPN-NEAT can hit the upper bound of the task score. Without the guidance of the curriculum, CuCo-NCU performs poorly on most tasks since learning to design and control from scratch in the high-dimensional design space is challenging.

We demonstrate the converged morphologies of all methods across three different runs and their corresponding performance in Pusher (Figure 4). For CuCo, we visualize the converged design of each stage. Direct-encoding methods like GA and BO prefer horse-like morphologies composed of randomly organized voxels. However, a randomly generated body may hinder coordinated behaviour. For instance, voxels in the last two robots created by GA often collide with each other, preventing robots from learning efficient control policies. CuCo-NCU fails to discover practical design and control policies with much lower performance. For indirect-encoding methods, both CuCo and CPPN-NEAT can discover homogeneous patches of materials like biological tissue. These patches are generated in a single pass by CPPN-NEAT, while CuCo develops these patches by module reuse, which can also be observed in many creatures, such as insects and vertebrates, which have several repeated body segments determined by homologous genes in different species Dellaert (1995). Interestingly, VSRs designed by CuCo obtain different skills for pushing a box. The first rhino-like robot chooses to push the box by its "horn". The second robot makes full use of the vertical actuators to let the box roll forward, while the third one uses horizontal actuators to contract itself and then pop the box out. In sum, CuCo holds great potential for exploiting design and control patterns learned in small spaces to produce high-performing robots in larger dimensional settings.

5.4 Ablations

We perform ablation studies to take a closer look at the behaviour of CuCo. Firstly, we demonstrate the learning curves of CuCo and CuCo-NCU (Figure 5), and we show that following an easy-to-difficult curriculum, it is much more efficient than learning from scratch in the target design space. Especially in Pusher, CuCo-NCU spends a large amount of time learning the connectivity. Secondly, we remove the self-attention mechanism used in our control policy and construct a new ablation **CuCo-NAT**. Evidently, by removing the self-attention mechanism, the control policy is unable to dynamically adapt to changes in morphologies, resulting in much lower performance. These two ablations indicate that the curriculum and the self-attention mechanism are indeed crucial for finding better solutions.

Another key feature of CuCo is the mechanism of inheritance. Between two learning stages, the converged morphology, the learned design and control policies from the last stage are inherited. We perform additional ablation studies to measure the impact of the inheritance on the subsequent learning process. We design the following ablations: (1) **CuCo-NIM**: No inheritance of the morphology from the last stage; (2) **CuCo-NID**: No inheritance of the design policy. We reset the parameters of the design policy at the end of each stage; (3) **CuCo-NIC**: No inheritance of the control policy. We reset the parameters of the control policy at the end of each stage; (4) **CuCo-NIDC**: No inheritance of both the design policy and the control policy. We reset all network parameters at the end of each stage. The learning curves of all ablations are shown in Figure 6. The solid curves represent the

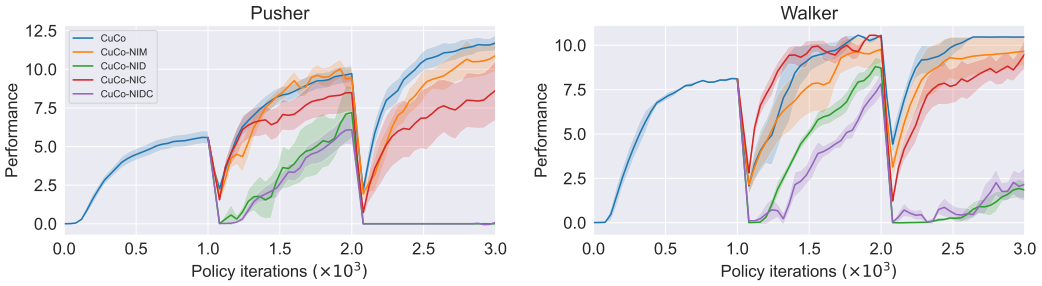

Figure 6: Ablations of the inheritance mechanism. The plots show that the inheritance of the design policy plays a critical role during the learning process.

mean values of the performance and the shaded regions indicate the standard deviations. For better visualization, we plot these curves from the second learning stage. It is worth noting that the inheritance of the design policy plays a critical role during the learning process, even if the control policy is reset, the performance is less affected, in comparison to CuCo-NID and CuCo-NIDC. The inheritance of the converged morphology also contributes to the learning process, as new morphologies can be quickly built by expanding the old morphology.

6 Discussion

We have shown that by following a predefined curriculum, we can learn to design and control VSRs through an easy-to-difficult process. We assume that we already had a large number of building blocks, and we do not constrain the number of modules for each type of material. Thus, the optimization is uncontrolled. An important direction of future work would be to investigate the constrained co-design method. For the generated morphologies, we can observe that our method less prefers empty voxels as shown in Figure 4, this is because we let the design policy learn the connectivity on its own, and the use of empty voxels may increase the risk of not meeting the connectivity. Compared with GA and BO, CuCo produces more plausible designs. Although some irregular morphologies can indeed help accomplish more complex tasks Bhatia et al. (2021), they are prohibitively expensive to obtain, in contrast, the main advantage of CuCo is the module reuse, which is more efficient for producing larger robot morphologies. Moreover, we use a simple NCA to encode the design patterns, there is indeed a need to explore more complex cell representations and network architectures. Although we found that CuCo performs well under the fixed curriculum, an important direction of future work is to explore more efficient curriculum learning methods.

There are two kinds of developments that happened at different timescales in CuCo. Within each stage, the robots grow under the restriction of body size. From one stage to the next, the robots will be bigger and stronger than the previous. However, size is not the only morphological characteristic of living creatures. An important line of exploration is to extend our method to help study the morphogenesis of virtual creatures and real-world embodied robots.

7 Conclusion

In this paper, we proposed CuCo, a curriculum-based co-design method for learning to design and control voxel-based soft robots through an easy-to-difficult process. Compared with previous approaches that learn from scratch in the target design space, our method is more efficient in producing larger robots with better performance as it follows a predefined curriculum and effectively reuses the knowledge of design and control obtained in small dimensional settings. Empirical evaluations on various robot control tasks show that CuCo outperforms prior methods in the final performance, producing intriguing robot designs with repeated body segments. In the future, we aim to extend our approach to more challenging tasks and transfer our method to physical shape-changing voxel-based modular robots.

REFERENCES

Dylan S. Banarse, Yoram Bachrach, Siqi Liu, Guy Lever, Nicolas Manfred Otto Heess, Chrisantha Fernando, Pushmeet Kohli, and Thore Graepel. The body is not a given: Joint agent policy learning and morphology evolution. In *AAMAS*, 2019.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML '09*, 2009.

Jagdeep Bhatia, Holly Jackson, Yunsheng Tian, Jie Xu, and Wojciech Matusik. Evolution gym: A large-scale benchmark for evolving soft robots. In *NeurIPS*, 2021.

Tianjian Chen, Zhanpeng He, and Matei T. Ciocarlie. Hardware as policy: Mechanical and computational co-optimization using deep reinforcement learning. In *CoRL*, 2020.

Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *GECCO '13*, 2013.

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

Frank Dellaert. Toward a biologically defensible model of development. 1995.

Kuan Fang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Discovering generalizable skills via automated generation of diverse tasks. *ArXiv*, abs/2106.13935, 2021.

Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature Communications*, 12, 2021.

Agrim Gupta, Linxi (Jim) Fan, Surya Ganguli, and Li Fei-Fei. Metamorph: Learning universal controllers with transformers. *ArXiv*, abs/2203.11931, 2022.

David R Ha. Reinforcement learning for improving agent design. *Artificial Life*, 25:352–365, 2019.

Aboul Ella Hassanien, Ajith Abraham, Athanasios V. Vasilakos, and Witold Pedrycz. Foundations of computational intelligence, volume 1: Learning and approximation. In *Foundations of Computational Intelligence*, 2009.

Donald J. Hejna, P. Abbeel, and Lerrel Pinto. Task-agnostic morphology evolution. *ArXiv*, abs/2102.13100, 2021.

Kazuya Horibe, Kathryn Walker, and Sebastian Risi. Regenerating soft robots through neural cellular automata. In *EuroGP*, 2021.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *ICML*, 2020.

Atil Iscen, George Yu, Alejandro Escontrela, Deepali Jain, Jie Tan, and Ken Caluwaerts. Learning agile locomotion skills with a mentor. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2019–2025, 2021.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff G. Schneider, and Barnabás Póczos. Parallelised bayesian optimisation via thompson sampling. In *AISTATS*, 2018.

Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *ArXiv*, abs/2010.01856, 2021.

Rich Gang Li, A. Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4051–4058, 2020.

Sida Liu, David Matthews, Sam Kriegman, and Josh C. Bongard. Voxcraft-sim, a gpu-accelerated voxel-based physics engine. 2020.

Kevin Sebastian Luck, Heni Ben Amor, and Roberto Calandra. Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning. In *CoRL*, 2019.

Eric Medvet, Alberto Bartoli, Federico Pigozzi, and Marco Rochelli. Biodiversity in evolved voxel-based soft robots. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2021.

Eric Medvet, Giorgia Nadizar, and Federico Pigozzi. On the impact of body material properties on neuroevolution for embodied agents: the case of voxel-based soft robots. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022.

A. Mordvintsev, E. Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020.

Tønnes F. Nygaard, David Howard, and Kyrre Glette. Real world morphological evolution is feasible. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020.

Rasmus Berg Palm, Miguel González Duque, Shyam Sudhakaran, and Sebastian Risi. Variational neural cellular automata. *ArXiv*, abs/2201.12360, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Deepak Pathak, Chris Lu, Trevor Darrell, Phillip Isola, and Alexei A. Efros. Learning to control self-assembling morphologies: A study of generalization via modularity. In *NeurIPS*, 2019.

Rolf Pfeifer and Josh C. Bongard. How the body shapes the way we think - a new view on intelligence. 2006.

Rolf Pfeifer, Fumiya Iida, and Max Lungarella. Cognition from the bottom up: on biological inspiration, body morphology, and soft materials. *Trends in Cognitive Sciences*, 18:404–413, 2014.

Federico Pigozzi, Yujin Tang, Eric Medvet, and David Ha. Evolving modular soft robots without explicit inter-module communication using local self-attention. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022.

Alvaro Sanchez-Gonzalez, Nicolas Manfred Otto Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter W. Battaglia. Graph networks as learnable physics engines for inference and control. *ArXiv*, abs/1806.01242, 2018.

Charles B. Schaff, David Yunis, Ayan Chakrabarti, and Matthew R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9798–9805, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Dylan S. Shah, Bilige Yang, Sam Kriegman, Michael Levin, Josh C. Bongard, and Rebecca Kramer-Bottiglio. Shape changing robots: Bioinspiration, simulation, and physical realization. *Advanced Materials*, 33, 2020.

Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial Life*, 1:353–372, 1994.

Andrew Spielberg, Allan Zhao, Yuanming Hu, Tao Du, Wojciech Matusik, and Daniela Rus. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. In *NeurIPS*, 2019.

Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8:131–162, 2007.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

Emma Hjellbrekke Stensby, Kai Olav Ellefsen, and Kyrre Glette. Co-optimising robot morphology and controller in a simulated open-ended environment. *ArXiv*, abs/2104.03062, 2021.

Shyam Sudhakaran, Elias Najarro, and Sebastian Risi. Goal-guided neural cellular automata: Learning to control self-organising systems. *ArXiv*, abs/2205.06806, 2022.

Jacopo Talamini, Eric Medvet, and Stefano Nichele. Criticality-driven evolution of adaptable morphologies of voxel-based soft-robots. *Frontiers in Robotics and AI*, 8, 2021.

Brandon Trabucco, Mariano Phielipp, and Glen Berseth. Anymorph: Learning transferable polices by inferring agent morphology. In *ICML*, 2022.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

Kathryn Walker and Helmut Hauser. Evolution of morphology through sculpting in a voxel based robot. In *ALIFE*, 2021.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Poet: open-ended coevolution of environments and their optimized solutions. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019a.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *ICML*, 2020.

Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Towards efficient automatic robot design. *ArXiv*, abs/1906.05370, 2019b.

Ye Yuan, Yuda Song, Zhengyi Luo, Wen Sun, and Kris M. Kitani. Transform2act: Learning a transform-and-control policy for efficient agent design. *ArXiv*, abs/2110.03659, 2022.

A ENVIRONMENT DETAILS

In this section, we provide details about the four tasks simulated in Evolution Gym platform Banarse et al. (2019).

Position. p^o is a 2-dim vector that represents the position of the center of mass of an object o in the simulation at time t. p^o_x and p^o_y are x and y components of this vector, respectively. p^o is calculated by averaging the positions of all the point-masses that make up object o at time t.

Velocity. v^o is a 2-dim vector that represents the velocity of the center of mass of an object o in the simulation at time t. v^o_x and v^o_y are x and y components of this vector, respectively. v^o is calculated by averaging the velocities of all the point-masses that make up object o at time t.

Orientation. θ^o is a 1-dim vector that represents the orientation of an object o in the simulation at time t. Let p_i be the position of point mass i of object o. θ^o is computed by averaging over all i the angle between the vector $p_i - p^o$ at time t and time 0. This average is a weighted average weighted by $||p_i - p^o||$ at time 0.

Other observations. $h^o_b(d)$ is a vector of length $(2d+1)$ that describes elevation information around the robot below its center of mass. More specifically, for some integer $x \leq d$, the corresponding entry in vector $h^o_b(d)$ will be the highest point of the terrain which is less than p^o_y between a range of $[x, x+1]$ voxels from p^o_x in the x-direction.

A.1 WALKER

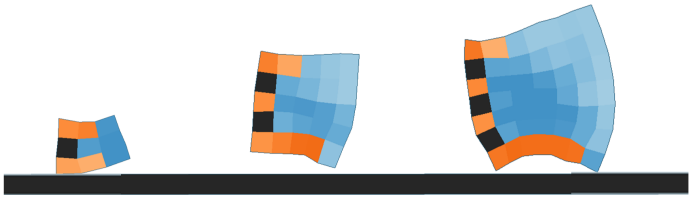

Figure 7: Walker environment.

In this task, the robot is rewarded by walking as far as possible on flat terrain. The target design space is 7×7. The task-specific observation is v^{robot}, and the reward R is:

$$R = \Delta p_x^{robot} \tag{1}$$

which rewards the robot for moving in the positive x-direction. The robot receives a reward of 1 for reaching the end of the terrain. The episode duration reaches a 500 timesteps.

A.2 UPSTEPPER

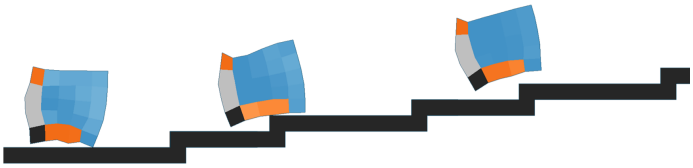

Figure 8: UpStepper environment.

In this task, the robot climbs up stairs of varying lengths. The target design space is 5×5. The task-specific observation is formed by concatenating vectors $\{v^{robot}, \theta^{robot}, h^{robot}_b(5)\}$, and the reward R is:

$$R = \Delta p_x^{robot} \tag{2}$$

which rewards the robot for moving in the positive x-direction. The robot also receives a one-time reward of 2 for reaching the end of the terrain, and a one-time penalty of -3 for rotating more than

75 degrees from its originally orientation in either direction (after which the environment resets). The episode duration reaches a 600 timesteps.

A.3 PUSHER

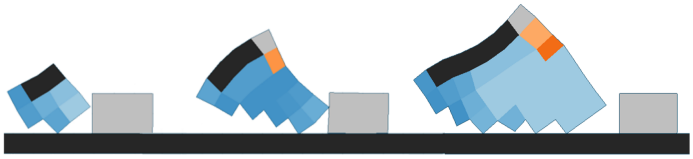

Figure 9: Pusher environment.

In this task, the robot pushes a box initialized in front of it. The target design space is 7×7. The task-specific observation is formed by concatenating vectors $\{v^{block}, p^{block} - p^{robot}, v^{robot}\}$, and the reward $R = R_1 + R_2$, where R_1 is:

$$R_1 = 0.5 \cdot \Delta p_x^{robot} + 0.75 \cdot \Delta p_x^{block} \tag{3}$$

which rewards the robot and box for moving in the positive x-direction, and R_2 is:

$$R_2 = -\Delta |p_x^{block} - p_x^{robot}| \tag{4}$$

which penalizes the robot and box for separating in the x-direction. The robot also receives a one-time reward of 1 for reaching the end of the terrain. The episode duration reaches a 500 timesteps.

A.4 JUMPER

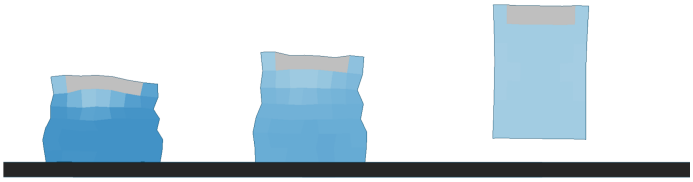

Figure 10: Jumper environment.

In this task, the robot jumps as high as possible in place on flat terrain. The target design space is 7×7. The task-specific observation is formed by concatenating vectors $\{v^{robot}, h_b^{robot}(2)\}$, and the reward R is:

$$R = 10 \cdot \Delta p_y^{robot} - 5 \cdot |\Delta p_x^{robot}| \tag{5}$$

which rewards the robot for moving in the positive y-direction and penalizes the robot for any motion in the x-direction. The episode duration reaches a 500 timesteps.

B VISUALIZATION OF MORPHOLOGY CHANGES

Here we provide additional visualization results of converged morphologies in other environments constructed using CuCo and other baselines (Figure 11). We also demonstrate CuCo's learning process in UpStepper and Pusher (Figure 12).

C IMPLEMENTATION DETAILS

We use PyTorch Paszke et al. (2019) to implement our proposed method in this work. We took the official implementation of transformer from Pytorch and added the position embedding. All hyperparameters for CuCo are listed in Table 2. For baseline algorithms, we use the official implementations of GA, CPPN-NEAT and BO in Evolution Gym platform Bhatia et al. (2021). We employ a population with a size of 12 for all evolution-based methods. Following the convention in

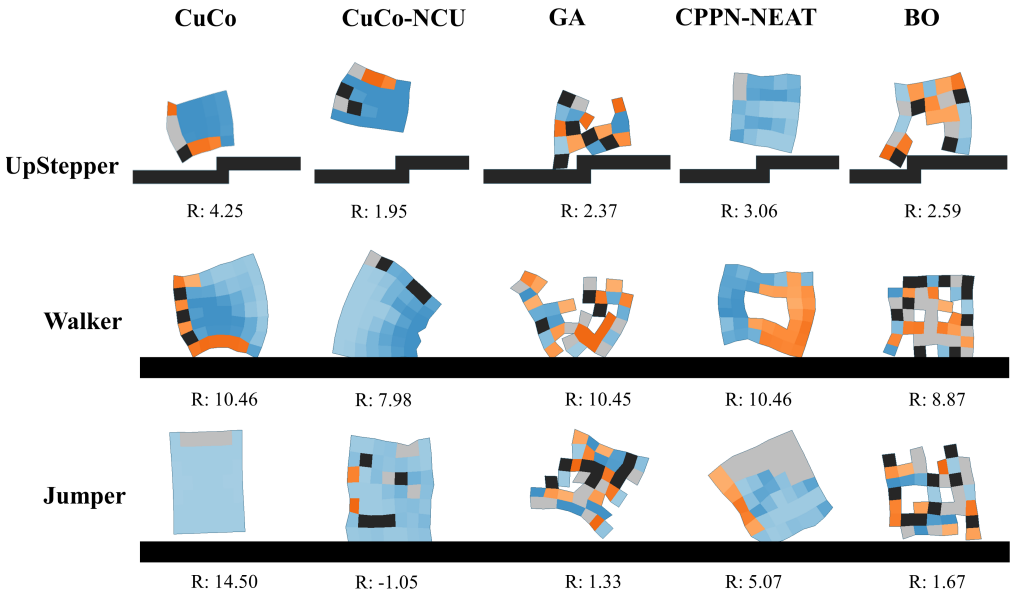

Figure 11: Visualization of converged morphologies of all methods in UpStepper, Walker and Jumper. Below each VSR is its average performance over five tests.

the literature, we use multiple CPU threads to accelerate the training of PPO. For all the environments used in the paper, it takes around 1 day to train our model on a computer with 12 CPU cores and an NVIDIA RTX 3090 GPU.

D Attention Matrices Analysis

Self-attention brings better interpretability than multilayer perceptron. In this section, we analyze the attention matrices that CuCo learns. We use only one transformer encoder layer in our work, thus we visualize the attention matrix after the input state passes through the attention layer. Figure 13 demonstrates the attention matrices of the critic network during the design process. Voxel 9 is assigned with more attention scores when the robot is being created. Figure 14 demonstrates attention matrices of the policy network at two execution steps. The color of each attention score tells the strength of the compatibility between inputs and interprets what is driving the behavior of the VSR. Evidently, when the robot's front foot (voxel 9) or the rear foot (voxel 7) touches the ground, the corresponding voxels are assigned with greater wights, which is consistent with our intuition and common sense.

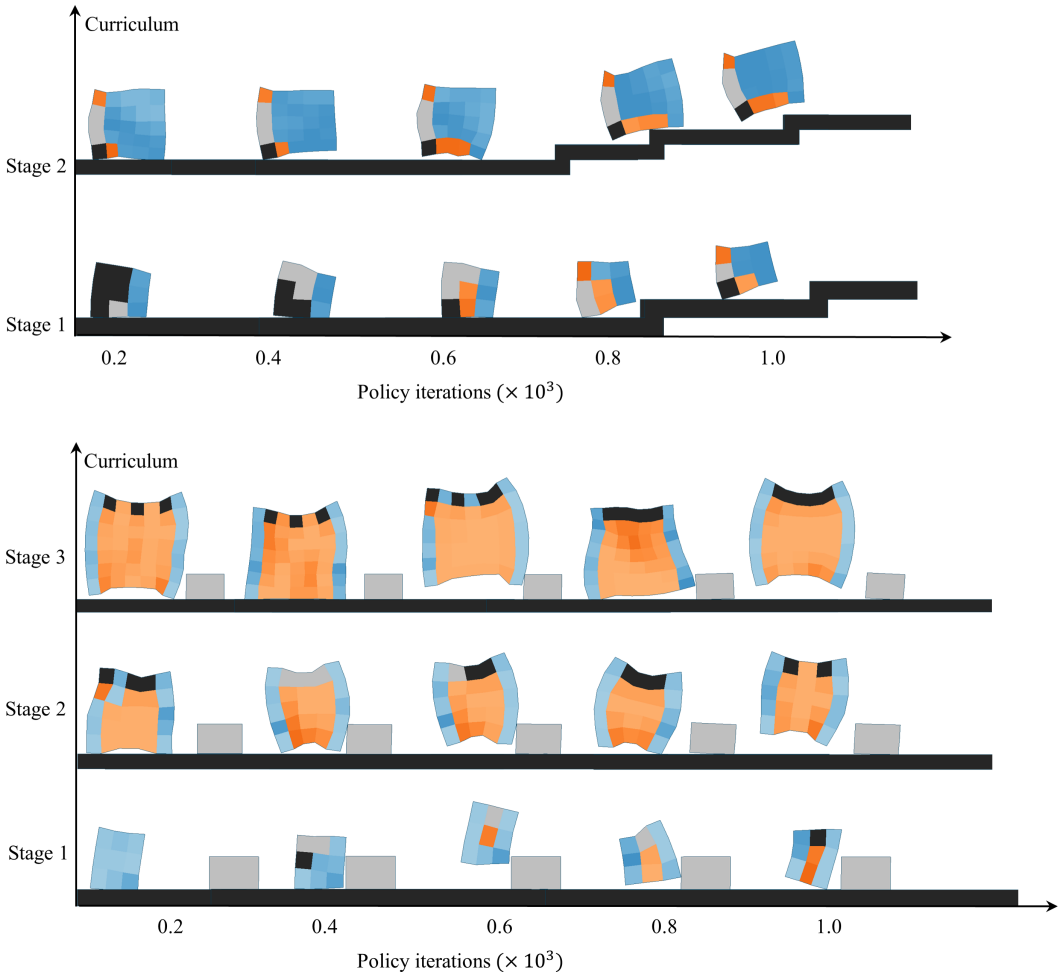

Figure 12: Visualization of CuCo's learning process in UpStepper and Pusher. The x-axis represents the number of policy iterations and the y-axis represents the curriculum. We show morphologies in the stage from left to right.

Table 2: Hyperparameters of CuCo.

	Hyperparameter	**Value**
PPO	Use GAE	True
	GAE parameter λ	0.95
	Learning rate	$2.5 \cdot 10^{-4}$
	Use linear learning rate decay	True
	Clip parameter	0.1
	Value loss coefficient	0.5
	Entropy coefficient	0.01
	Timesteps per rollout	2048
	Num processes	4
	Optimizer	Adam
	Evaluation interval	10
	Discount factor γ	0.99
	Clipped value function	True
	Observation normalization	True
	Observation clipping	$[-10, 10]$
	Reward normalization	True
	Reward clipping	$[-10, 10]$
	Policy epochs	8
	Policy iterations per stage	1000
NCA	Neighborhood	Moore
	Input dimension	9
	Output dimension	5
	Growth factor	False
	Hidden layers	$(64, 64)$
	Activation function	Tanh
	Design steps	10
Transformer	Number of layers	1
	Number of attention heads	1
	Embedding dimension	64
	Feedforward dimension	128
	Non linearity function	ReLU
	Dropout	0.0

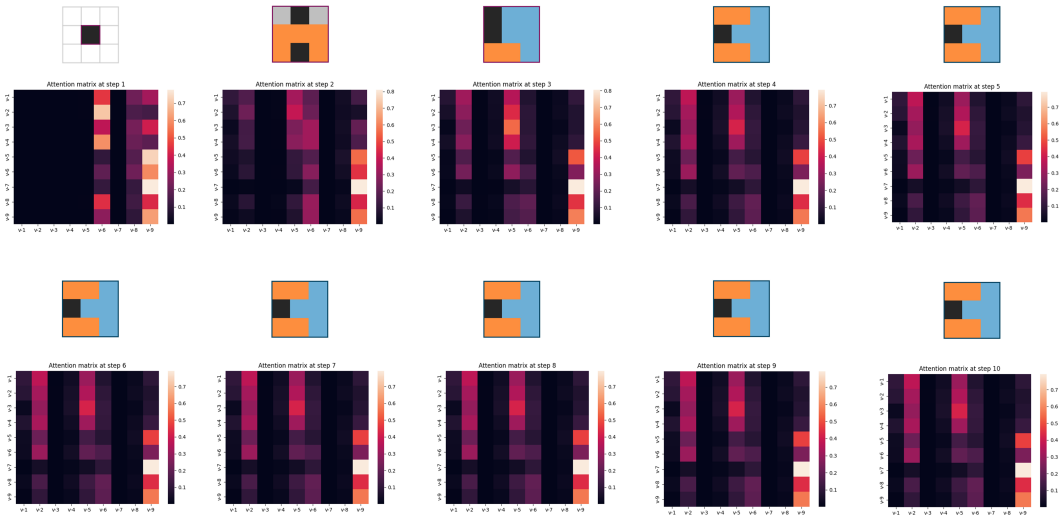

Figure 13: Attention matrices of 10 design steps that are produced by the critic network.

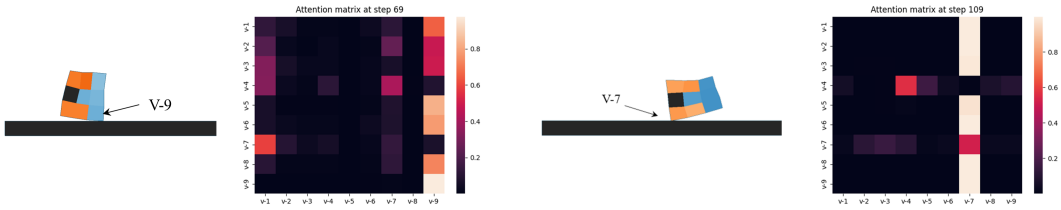

Figure 14: Attention matrices of 2 control steps that are produced by the policy network.