國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

利用視覺語言模型生成與現實對應的訓練環境課程以提升具物理泛化能力的控制策略

Improving Physics-Based Control with Grounded Environment Curriculum Generation via Vision-Language Models

周玉鑫

Yu-Hsin Chou

指導教授: 林軒田 博士

Advisor: Hsuan-Tien Lin Ph.D.

中華民國 114 年 8 月

August, 2025

doi:10.6342/NTU202501834

Acknowledgements

I am deeply grateful to my advisor, Prof. Hsuan-Tien Lin. Our weekly research meetings have played a pivotal role in shaping how I think and build stronger arguments on research ideas. You often pointed out my blind spots while giving me the freedom to explore possible solutions and explanations. Your guidance made these two years truly worthwhile, and I will always remember how you challenged my thoughts and helped me strengthen my ideas. The lessons I've learned from you will stay with me moving forward.

I also appreciate the support from my labmates. Over the past two years, we've shared many discussions about research ideas, course projects, and life beyond academics. Your insights and encouragement were invaluable throughout this journey.

I would like to thank my oral exam committee for their critical feedback and suggestions. Your input helped make this work more rigorous and complete, and I am truly grateful for your time and effort.

I am sincerely thankful to my parents for their unwavering support and encouragement. They have always believed in my choices and encouraged me to explore the future I envision. I would not be who I am today without the space they gave me to follow my interests and pursue the things I love.

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

摘要

基於物理的控制任務需要具備良好的泛化能力，因為違反物理定律（例如重力、碰撞等）可能帶來嚴重的安全風險。我們探討如何透過產生訓練環境課程來提升在此類任務的泛化能力。基於無監督環境設計框架，我們發現既有方法中所採用的隨機環境產生器，可能削弱零樣本泛化能力。透過檢查其產生的環境，我們發現這些產生的環境往往過於複雜。為了解決這個問題，因為視覺語言模型無需額外訓練，且可在零樣本的情況使用與進行條件化控制，我們利用隨即可用的視覺語言模型來產生與現實對應的訓練環境。我們進一步以語意對應性與樣本複雜度對應性兩項指標，衡量所產生的環境與參考環境及策略的對應性，並提出多項重要的設計決策以提升這兩個指標。實驗結果顯示，即便僅使用具現實對應的環境產生器，就能顯著提升泛化能力，並可透過結合互補的無監督環境設計方法來進一步增強。我們提出的方法 V-SFL，在所研究的基於物理的控制任務中達到最佳表現。

關鍵字：無監督環境生成、視覺語言模型、強化學習、基於物理的控制

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

Abstract

Physics-based control tasks demand robust generalization because violations of physical laws, such as those involving gravity or collisions, could cause severe safety risks. We investigate how to improve generalization in such tasks by generating a training environment curriculum. Building on the framework of Unsupervised Environment Design (UED), we identify that random environment generators, as adopted by several prior UED works, could hinder zero-shot generalization. By examining the generated environments, we found that the generated environments are often overly complex. To address this, we use off-the-shelf Vision-Language Models (VLMs) to produce environments with grounded complexity, leveraging that VLMs are training-free and can be conditioned in a zero-shot manner. We further define grounded complexity by semantic groundedness and sample complexity groundedness to reflect how grounded the generated environments are with respect to a reference environment and policy. We outline several design choices to achieve these metrics. Experimental results demonstrate that even a grounded environ-

doi:10.6342/NTU202501834

ment generator alone improves generalization. Furthermore, performance can be further boosted by incorporating a complementary UED method. Our proposed method, **V**LM-based **S**ampling **F**or **L**earnability (**V-SFL**), achieves state-of-the-art performance on the studied physics-based control benchmark.

Keywords: Unsupervised Environment Design, Vision-Language Models, Reinforcement Learning, Physics-based control

doi:10.6342/NTU202501834

Contents

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

List of Figures

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

List of Tables

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

Denotation

RL	強化學習 (Reinforcement Learning)
UED	無監督環境設計 (Unsupervised Environment Design)
VLM	視覺語言模型 (Vision-Language Models)

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

Chapter 1 Introduction

Recent developments of generation models, such as Vision-Language Models (VLM) and video generation models, have brought impressive capabilities in world understanding and prediction. However, such capabilities are still imperfect, as sometimes the model output could violate physical laws, making the model less trustworthy. This urges researchers to study the understanding of physical laws in various ways, such as visual-question answering [7, 39], object-centric models [3, 22, 24, 37, 38], world model physics alignment [20], self-supervised objectives [14].

We focus on physics-based *control* generalization because violations of physical laws can lead to severe safety risks, such as collisions, which are more critical than errors in video generation or question answering. Specifically, we aim to improve physics-based control generalization with gravity and collision on a variety of object layouts, as gravity and collision are the most common scenarios of physics-based control. The control objective is for the subject to reach a goal location by rolling on the surfaces of other objects under gravity. We train control policies in a set of training environments and test whether the trained policy could perform well in unseen testing environments, in both zero-shot manner and after a period of adaptation in the unseen environments. For example, policy is trained in environments with a set of possible steepness of slope, and we test whether the policy could still work when the steepness of slope is unseen before. Moreover, there may

doi:10.6342/NTU202501834

be obstacles on the subject's trajectory, and the position of the obstacle could be different in training and testing environments. This leads to different trajectories after collision, and the policy should be generalizable to such unseen environments.

We conduct policy learning via online model-free reinforcement learning due to the following reasons. First, offline methods such as imitation learning or offline RL [17] require a dataset of trajectories. Such trajectories can be collected using algorithms, existing policy models, or expert demonstration [12]. However, in physics-based control, it is unclear by which algorithm or model the trajectories should be produced, and expert demonstration is too costly. Therefore, we choose to train the policy online. Second, model-based methods, such as a pre-trained video model or a differentiable physics-based simulator, can create another source of error compared to model-free methods. Video models could produce predictions that violate physical laws [20], and the gradient obtained from differentiable simulators is unstable and requires special design to mitigate the problem [26, 30]. Therefore, we decide to train the policy in a model-free manner.

In the online model-free RL setup, to promote generalization, a widely adopted method is domain randomization (DR) [32]. DR generates a large amount of environments to train the control policy, hoping the training environments could be broad enough to cover the possible environments at test time. Various techniques have been proposed for better DR [8, 40]. However, in physics-based control, a change in slope steepness or obstacle location could make environments significantly harder, because it is difficult to sample a successful trajectory due to the lack of dense reward. We assume the reward is sparse due to the difficulty in designing the reward function. In this way, not all environments offer the same learning utility, and further training in difficult environments could also lead to forgetting [36]. To address those problems, we opt for Unsupervised Environment

doi:10.6342/NTU202501834

Design (UED) methods because they try to quantify the learning utility of environments. The learning utility is typically quantified by regret, corresponding to the current control policy. The goal of UED is to construct a curriculum of environments that maximizes the generalization performance to unseen environments. In the seminal work of UED [11], it has been shown that training policies in high-regret environments could facilitate generalization.

An important research obstacle for studying the generalization is choosing a set of benchmarks to evaluate on. We think that the benchmark should satisfy the following properties: controllability, discrete action, sparse reward, and texture-free observation. Controllability refers to the ease of manipulating environments through variables or programs, a key requirement for UED. For instance, well-known Atari [4] does not offer such controllability. In addition, discrete action tasks are more preferable than continuous action tasks (such as RLBench [16]) because continuous action tasks require the policy to learn motor control, which could add another layer of complexity. Furthermore, the sparse reward property is important because the reward function is hard to design and an ill-defined reward function could result in a form of bias. Therefore, we think that the performance should be assessed through whether the task is accomplished. Finally, we focus on tasks with texture-free observation because the policy could focus directly on learning physical law generalization, where we note that various works have discussed tasks with rich textures [40]. Of several potential benchmarks, we found that the recently proposed benchmark I-PHYRE [22] satisfies all the properties, and it will serve as our main testbed for evaluating physics-based control generalization.

UED methods could be classified as learning-based methods and replay-based methods. Learning-based methods generate environments via a learned generator, whereas

doi:10.6342/NTU202501834

replay-based methods prioritize environments sampled from a random generator. Learning-based methods are typically hard to train [10] or require a pre-defined dataset [10, 13]. However, such a dataset is not available in our setup. Therefore, we applied existing replay-based methods [19, 27, 28] but found that training in randomly-generated environments does not lead to an improved generalization. Inspecting the generated environments, we identified that the generator often produces overly complex environments. Such a complexity could hinder policy learning and, counterintuitively, we do not observe meaningful performance differences with replay-based UED variants.

Observing the phenomenon, we ask: ”*Can grounded environment curriculum generation improve generalization?*” Because of the lack of predefined dataset and the need to avoid high complexity, a natural approach to this problem is to leverage the capabilities of off-the-shelf VLMs. VLMs are training-free and can be conditioned in a zero-shot manner [6, 21], making them an immediate choice to ground the complexity. However, we found that a direct use of a VLM to generate environments results in very random outputs, producing trivial environments solvable by a random policy. Based on this observation, we identify two essential components of ”grounded complexity”: semantic groundedness and sample complexity groundedness. Semantic groundedness measures similarity to a reference environment, while sample complexity groundedness measures how many samples a reference policy needs for success. Those metrics prevent the generated environments from being overly complex from both semantic and sample complexity perspectives. We then study how to elicit the VLM to enhance both metrics. Experimental results show that VLM, when properly grounded, serves as a promising generator that captures environment complexity and outperforms existing UED baselines. Furthermore, our grounded environment generator is compatible with replay-based UED methods, and we show that

doi:10.6342/NTU202501834

their integration, V-SFL, reaches state-of-the-art performance in the studied physics-based control benchmark.

Our work made the following contributions. First, we first define grounded complexity under the UED regime and demonstrate how to elicit such groundedness from VLM. Second, we show that grounded environment curriculum generation is crucial for generalization for physics-based control.

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

Chapter 2 Related works

Physics-based control and intuitive physics The capability for physical understanding has been studied across a variety of domains, including Visual Question Answering (VQA) [7, 39], video prediction [2, 3, 24], object-centric representation [37, 38], world models [20], self-supervised training objectives [14]. While most of these works emphasize semantic or predictive understanding, few have investigated the control perspective in an isolated manner. The most relevant to our work are [22] and [25], which study generalization behavior in physics-based control environments. [25] focus on scaling up physical simulations for more efficient training, whereas our work focuses on the central role of the environment generator. Moreover, the benchmark proposed by [22] presents limitations for evaluating UED methods, as its small number of test environments hinders a meaningful comparison.

LLM/VLM-based task generation Large language and vision-language models have recently been leveraged to produce diverse robotic simulation tasks at scale. Various modalities have been explored for different input and output, including text-based interfaces [34, 35], image-based prompts [41], and intermediate representations [31]. Some works aim to improve real-world transferability by generating simulation scenes tailored to real-world utility [40]. While many of these studies demonstrate the effectiveness of

doi:10.6342/NTU202501834

task generation by showing downstream policy improvement, they did not examine the alignment of agent's capability with the generated environments. The most relevant to our work is [23], which improves VLM-based terrain generation by rolling out multiple competing policies in an evolutionary manner. In contrast, our work focuses on the grounded complexity of environment generation and evaluates on an isolated physics-based control benchmark.

Unsupervised Environment Design Unsupervised Environment Design (UED) aims to improve generalization in RL by strategically generating a training environment curriculum without supervision. The line of research began with [11], and subsequent methods have improved generalization through alternative regret approximations or additional components in UED. UED methods could be categorized into learning-based methods and replay-based methods. Learning-based methods train an adversary that generates high-regret environments via reinforcement learning [11, 19], auxiliary representation learning [1], VAE [13], and diffusion models [10]. Replay-based methods, on the other hand, approximate the regret of existing or randomly generated environments, such as ℓ_1 value loss [19, 27], zero-shot success rate [28], and others [5]. Learning-based methods are often difficult to train [10], or require an additional dataset [10, 13], making those method inapplicable to setups with high-dimensional observation space due to the lack of a dataset like ours. Replay-based methods, on the other hand, assume a random environment generator, which turns out to be unrealistic beyond simple benchmarks like MiniGrid [9], BipedalWalker, and CarRacing [33].

doi:10.6342/NTU202501834

Chapter 3 Background

Reinforcement Learning In reinforcement learning, a POMDP environment could be modeled by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{R}, \mathcal{I}, \gamma \rangle$ where $\mathcal{S}, \mathcal{A}, \mathcal{O}$ means state, action, and observation, respectively. γ is the discount factor. The environment transition is described by $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \triangle(\mathcal{O})$, the reward function is modeled by $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and the observation function is given by $\mathcal{I} : \mathcal{S} \to \mathcal{O}$. We consider the episodic setup where we assume that the episodic duration is T. The goal of reinforcement learning given $\mathcal{M}$ is to learn a control policy $\pi \in \Pi$ that maximizes the expectation of discounted reward $E_\pi[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$.

Unsupervised Environment Design Following [11], the POMDP can be extended to an under-specific POMDP (UPOMDP) given by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{R}, \mathcal{I}, \gamma, \Theta \rangle$ where the extra Θ implies a distribution over all possible environments. In this setup, the transition function becomes $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \Theta \to \triangle(\mathcal{S})$. The goal of UED is to generate a curriculum of environment during the course of learning and maximize the generalization performance (often, zero-shot) of the learned policy in a set of unseen environments $\mathcal{M}_{\text{test}} \subset \Theta$, while training in a set of designated environments $\theta \in \mathcal{M}_{\text{train}} \subset \Theta$. An environment generator (or adversary in some prior work) $\mathcal{G} : \Pi \to \Theta$ is used to randomly generate an environment given a policy and can be parameterized by a program or a learned model. Within the

doi:10.6342/NTU202501834

generator, we typically prioritize environments that provide better learning opportunities defined by $\text{Regret}_{\pi}(\theta)$ given the current policy π.

doi:10.6342/NTU202501834

Chapter 4 Methodology

The central idea of our method is to generate environments with grounded complexity. In Section 4.1, we demonstrate the problem of a naive use of VLM as an environment generator and define the meaning of grounded complexity. To solve the problem, we make several design choices in Section 4.2 to activate grounded environment generation. Then, we show how to integrate a varied replay-based method to prioritize environments with high regret in Section 4.3. Finally, we introduce the testing environments that we designed in Section 4.4. For an overview of our method, see Figure 4.1.

4.1 Why do we need grounded complexity?

We first attempt to understand why we need the generated environments with grounded complexity by empirical observations. First, we generate environments by a naive way. To generate, because the block configuration space is quite large, as each block has two endpoints lying in a 600x600 canva, a reasonable choice is to leverage an off-the-shelf VLM to generate the environments, assuming that VLMs are trained on a large corpus of human knowledge and have been proven useful in robotic task generation. We note that generating environments by human is time-consuming and not scalable. We query the VLM with the game description, the designate output format, and we set the goal to

doi:10.6342/NTU202501834

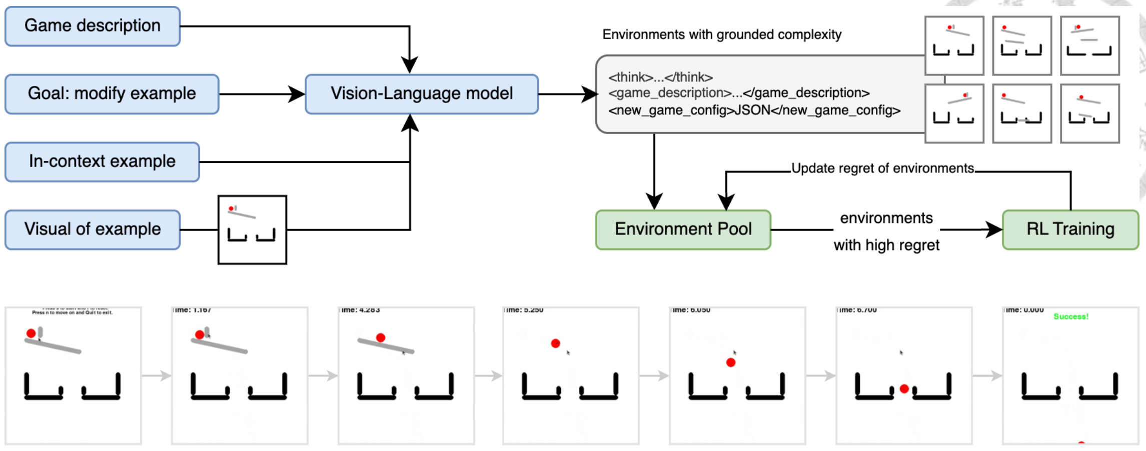

Game description
Goal: modify example
In-context example
Visual of example
Vision-Language model
Environments with grounded complexity
<think>...</think>
<game_description>...</game_description>
<new_game_config>JSON</new_game_config>
Update regret of environments
Environment Pool
environments with high regret
RL Training
Success!

Figure 4.1: **Overview of the proposed method (V-SFL)**. **Top**: To make generated environments with grounded complexity, we use a VLM with in-context example, its visualization and properly prompt the VLM with game description and set the goal to be modifying the example config. The generated environments are in JSON format. Those environments will be used to train RL policy and will be prioritized by the regret value computed every I iteration. **Bottom**: A successful gameplay of the in-context example.

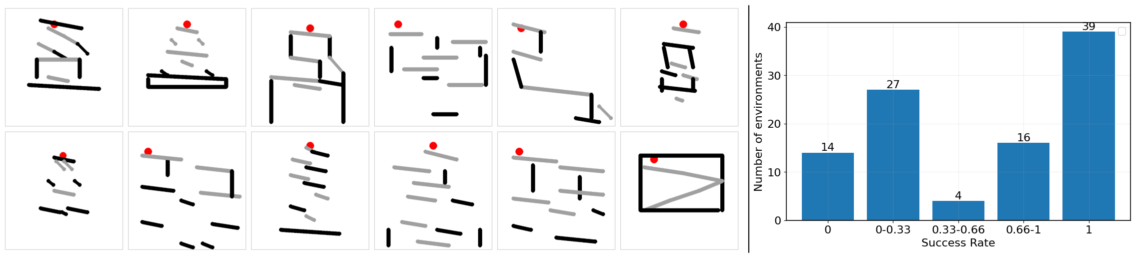


Figure 4.2: **Why grounded complexity is important**: Naive application of VLM could result to environments that could be solved by random actions but overly complicated semantic layouts. **Left**: the visualization of generated environments. **Right**: the success rate on the environments by a random policy is close to 100%, demonstrating the problem of ungrounded sample complexity.

"generate a new game config" (in JSON). Meanwhile, we avoid giving instructions on how the environment should be generated to keep as unsupervised as possible.

We generated a total of 100 environments by GPT-4.1, and the JSON results are visualized in Figure 4.2 (left). We observe some problems here. First, the generated environments are semantically unaligned with the goal of the game because many gray blocks (which can be eliminated) are unrelated to the possible trajectories of the red ball. Moreover, it is semantically overly complicated from human perspective. Humans are unlikely to design such complicated layouts at first. To understand this point further, for

doi:10.6342/NTU202501834

each generated environment, we roll out a random policy 500 times and count the percentage of successes over all rollouts. From Figure 4.2 (right), we found that 39 out of 100 environments could be succeeded with 100% success rate by a random policy. Such environments are undesirable for policy learning because many environments could be arbitrarily succeeded by a random policy. Consequently, we argue that an ideal generator should generate environments with "grounded complexity", defined by: (1) **Semantic groundedness**. This means that the generated environments should preserve semantically similar, but not too similar to a reference example environment. We will evaluate this metric by cosine similarity on CLIP embeddings. (2) **Sample complexity groundedness**. This means that by rolling out trajectories by a reference policy (in our work, we use a random policy), the success rate should not be close to 100% or 0%. Our protocol to determine the method with best groundedness is by eliminating extreme values, reflecting the idea that the environments generated can neither be too complex or overly grounded.

4.2 Grounded environment generation with VLM

To ensure grounded complexity, we made several important design choices. First, we revised the prompt in Section 4.1 to be "Generate a new game config by modifying the given game config as provided below." See Appendix for the complete prompt. This leverages the in-context learning capabilities inherent in VLMs. The example game configuration corresponds to a layout which there is a hole below and the red ball rolls on a tilted block that can be eliminated. This comes with adequate sample complexity, and, therefore, the generation from the VLM is more likely to inherit such a property. Secondly, to ensure that the generation is reliable and adheres to the example configuration with high probability, we add the visualization of the example game as an additional visual

doi:10.6342/NTU202501834

input. Furthermore, to make the VLM produce more diverse results, we force the VLM to think before producing the result. Finally, we pre-generate a huge pool of environments $\mathcal{P}$ with size N because if we generate environments during the training progress, it tends to block the training due to the time it takes to generate new environments. Concretely, we generate N environments using a VLM-based generator $\mathcal{G}$ with an in-context example:

$$\mathcal{P} \leftarrow \theta_1, \theta_2, \ldots, \theta_N \sim \mathcal{G}(\text{in-context example with visualization})$$

4.3 Learnability-based regret approximation

With a pre-generate pool $\mathcal{P}$ with many environments, we integrate a learnability-based regret approximation to select the environments with high regret for policy learning. We follow the concept of learnability from [28] but made an extension. Let p denote the zero-shot success rate by the current policy for an environment. [28] uses $p(1-p)$ to approximate the regret for the environment. In contrast, we use a Beta distribution with a hyperparameter α to approximate the regret because we empirically found that α could non-trivially affect the generalization within specific environment steps. Concretely, the regret of an environment is defined by

$$P_\pi(\theta, \alpha) \propto (p)^\alpha (1-p)^{1-\alpha}, 0 < \alpha < 1, \theta \in \mathcal{B}$$

Therefore, we sample environments following the distribution of $P_\pi(\theta, \alpha)$. The approximation reflects that environments with a zero-shot success rate close to α will be associated with higher regret. When α is 0.5, it recovers the method of [28]. Because the pool $\mathcal{P}$ could be arbitrarily large, evaluating the zero-shot success rate in all environments would

doi:10.6342/NTU202501834

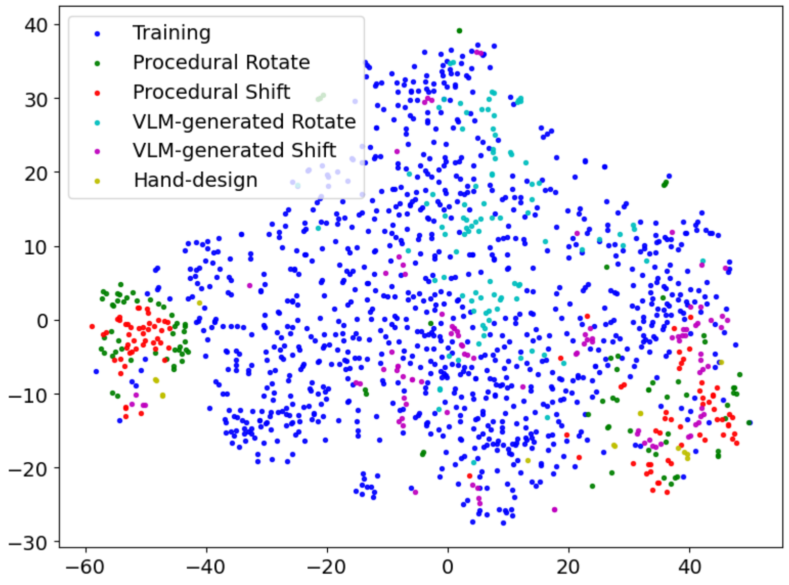


Figure 4.3: t-SNE visualization of the training and testing environment sets

require expensive computation. Therefore, we evaluate the regret value every I iteration and for each evaluation, we first subsample the pool $\mathcal{P}$ to a buffer $\mathcal{B}$ with $N' = 1000$ environments. Furthermore, to avoid stale environments, following [19], we also sample environments uniformly with probability ρ. Indeed, our integration of learnability-based environment curriculum is not novel but is an important and simple ingredient to enhance generalization performance.

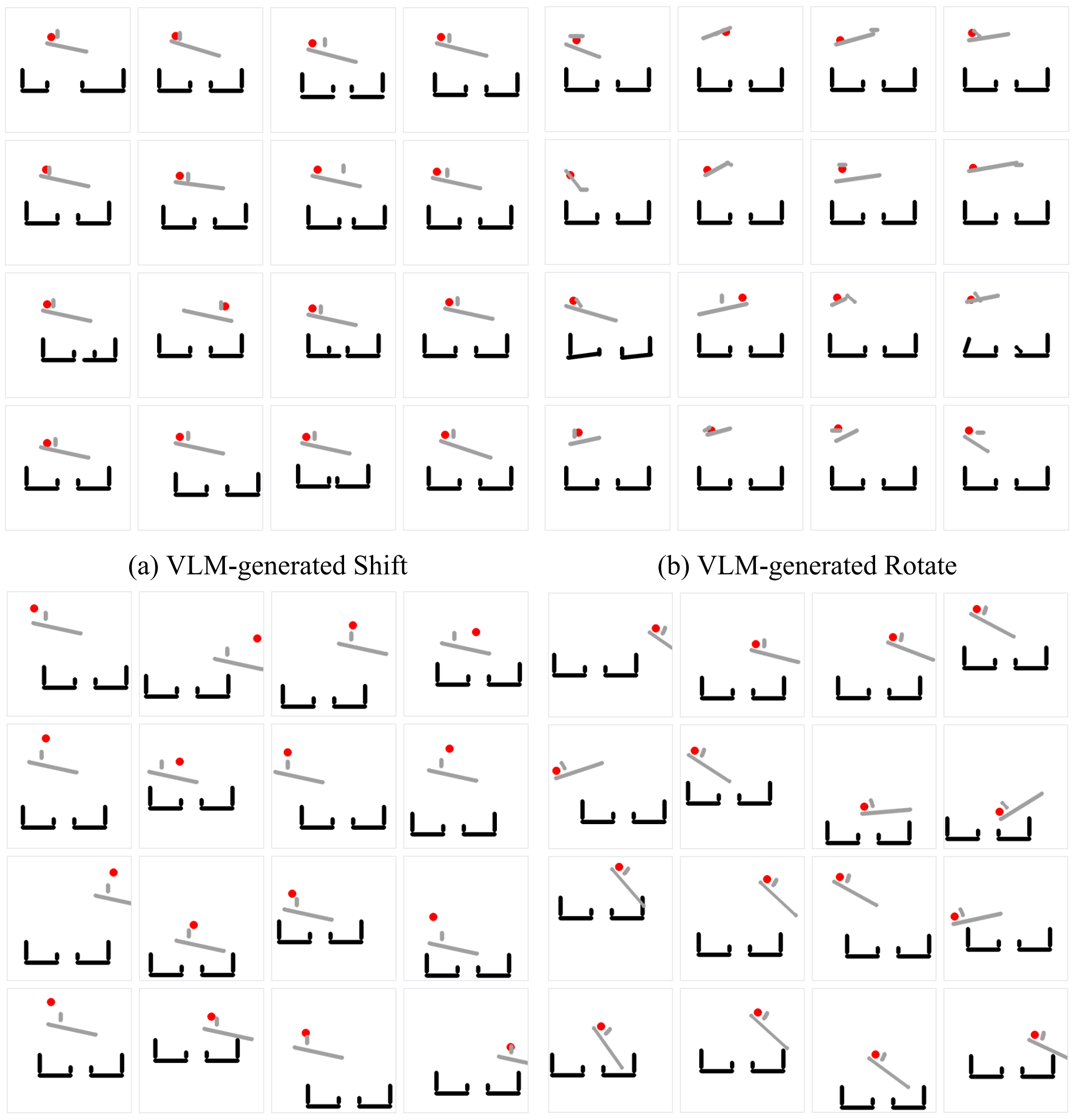
4.4 Testing environments for evaluating generalization performance

Previous work often evaluates the performance of UED methods in a set of unseen environments or several hand-designed environments. However, the benchmark [22] does not offer a set of environment for UED evaluation, so we generate the testing environments ourselves. We designed three kinds of test environments: VLM-generated environments (in-distribution), procedurally generated environments (out-of-distribution), and

doi:10.6342/NTU202501834

hand-design environments (adversarial). First, as our training environments are generated by VLM, using VLM to generate additional environments to test resembles the training environments more. We additionally prompt the VLM that it must generate new configurations by heuristic like rotating blocks and shifting blocks, leading to VLM-generated Rotate and VLM-generated Shift. Second, to produce environments that are distinct from the training distribution, we use a program for generation. In particular, we vary the blocks on the upper half and bottom half differently with offsets and rotations, resulting in Procedural Rotate and Procedural Shift. Finally, we make hand-designed environments as a total of 15 environments adversarially. Specifically, we investigate failure cases while rolling out a policy trained on the example and its variations. We collect the failure cases and note that those cases are still failure cases after we use a VLM-based generator to produce environments. We validate the distribution of testing environments compared to the training distribution by t-SNE on the CLIP embedding of the environment visualization (the initial scene). From Figure 4.3, we observe that the embedding of Procedural Rotate/ Shift forms a small group on the left and right of the figure, showing it is dissimilar to the training distribution. See Figure 4.4 for samples. For details, please refer to the Appendix.

doi:10.6342/NTU202501834



(a) VLM-generated Shift

(b) VLM-generated Rotate

(c) Procedural Shift

(d) Procedural Rotate

Figure 4.4: **Visualization of testing environments**. We show examples of VLM-generated environments and procedurally generated environments with shift and rotate variations.

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

Chapter 5 Experiments

To understand how the proposed method improves generalization in physics-based control, we address the following questions in our experiments: **(1)** How important is each design choice made in our method? **(2)** How is the proposed method compared to prior work on zero-shot performance? **(3)** How fast does the proposed method adapt to unseen environment(s)? **(4)** Can another VLM be used to score the environments and generate a curriculum based on the score? To ensure a fair comparison, in our experiments, we evaluated the success rate by rolling out 20 episodes in each environment, and we averaged the result over three runs with different random seeds.

5.1 How important is each design choice made in our method?

We justify the design choices made in our VLM-based environment generator through a series of analyses. First, we validate that these design choices lead to the generation of environments with grounded complexity. Secondly, to connect grounded complexity to generalization performance, we focus on the zero-shot generalization by training a policy on environments generated by different methods. Finally, we provide some qualitative samples.

For grounded complexity, we compare four methods: (1) **No variation**: the same

doi:10.6342/NTU202501834

Method	Mean Score	Std Dev	Min	Max
No variation	0.8632	0.0286	0.7740	0.9554
Variation	0.9512	0.0276	0.8607	0.9892
Variation + Visual	0.9714	0.0219	0.8732	0.9964
Variation + Visual + Think	0.9497	0.0360	0.8082	0.9923
Random	0.9386	0.0394	0.8314	1.0000

Table 5.1: **Semantic groundedness of environments on ablations**: Number indicates the cosine similarity of CLIP embedding between generated environments and the example.

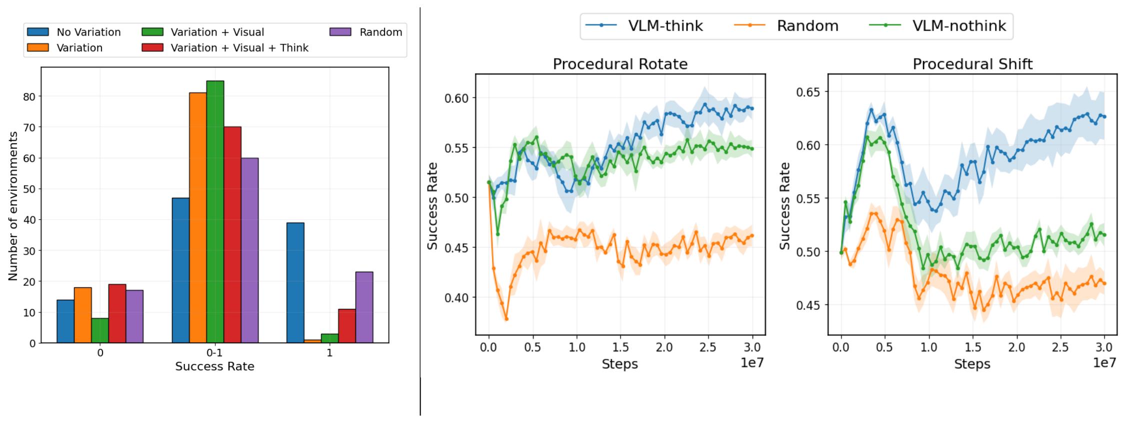


Figure 5.1: **Left**: Sample complexity groundedness. It is less grounded if the generated environments could achieve 100% success rate by random policy. **Right**: Zero-shot generalization performance by training on environments generated by different methods

setting as Section 4.1. (2) **Variation**: ask the VLM to modify the provided game config. (3) **Variation + Visual**: on the top of (2) Variation, we add the visualization to the input. (4) **Variation + Visual + Think**: on the top of (3) with thinking. (5) **Random**: a random generation procedure that modifies a provided game example, and we iterate each block and add an offset in $[-100, 100]$ with probability 0.3. We use the same protocol as Section 4.1 and roll out a random policy for 500 times on each environment. We generate 100 environments by each method. We evaluated both semantic groundedness and sample complexity groundedness metrics on each method. From Table 5.1, we conclude that with Variation and Visual component, the semantic groundedness could be improved, as seen from the mean score metric. With Think, while the mean score is slightly lower, its standard deviation is larger, meaning it generates more diverse results while staying grounded. From 5.1 (left), both No Variation and Random method generate trivial envi-

doi:10.6342/NTU202501834

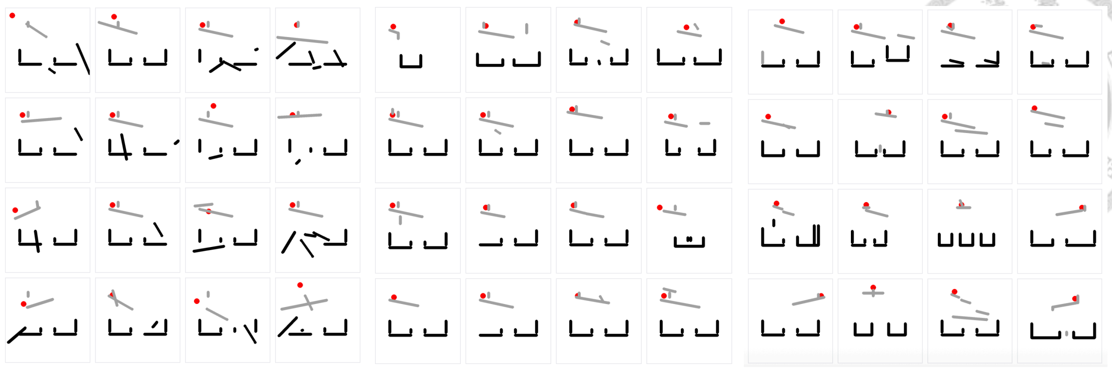

(a) Random generation (b) VLM-nothink (c) VLM-think

Figure 5.2: **Qualitative comparison between different environment generators**. Random generation often creates layouts that are not semantically grounded. VLM-think, compared to VLM-nothink, produces more diverse layouts that encourage generalization.

ronments solvable by random policies. Indeed, as we will see in the next experiment, a non-extreme result on both semantic groundedness and sample complexity groundedness could be more beneficial in generalization, as opposed to the best ones.

For generalization, we compare three types of environment generation: (1) **Random**: the Random method in the previous experiment and we scale to 4000 environments. (2) **VLM-nothink**: the Variation + Visual method and we scale to 4000 environments, (3) **VLM-think**: the Variation + Visual + Think and we scale to 4000 environments. We trained a PPO [29] policy based on the generated environments which are sampled uniformly. The result can be seen in Figure 5.1 (right). We confirmed that all design choices made are essential to yield better zero-shot generalization performance.

Figure 5.2 shows the visualization of the environments generated by different methods. VLM-think achieves the best generation quality as it produces more diverse layouts while keeping the complexity grounded. Notably, the red ball location is also more diverse in VLM-think generations. Interestingly, this echoes the finding from the literature of Large Language Models that zero-shot chain-of-thought is crucial in logical tasks such as math and coding [21]. Finally, the result of Random is the worst and the outputs are

doi:10.6342/NTU202501834

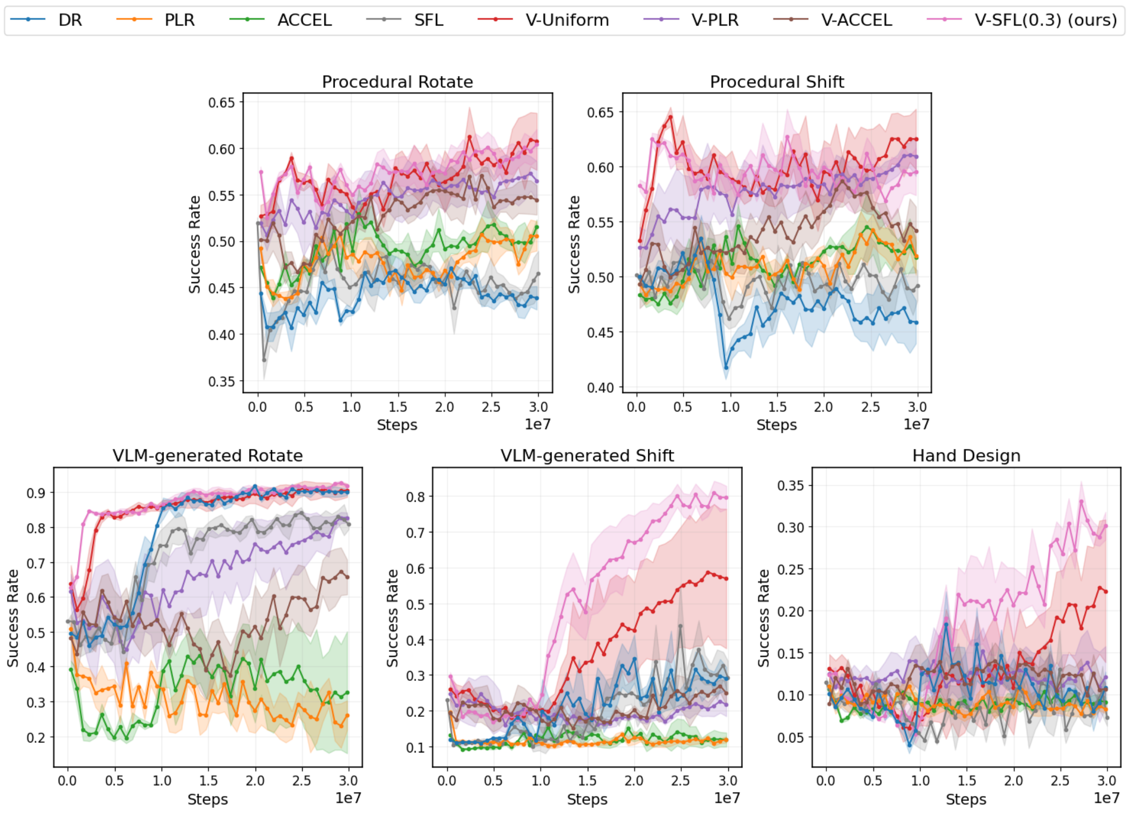


Figure 5.3: **Comparison between different UED methods.** Prior UED methods typically do not work well due to the assumption of a random environment generator. V-SFL/ V-PLR/V-Uniform could achieve substantial improvement on all test sets. Shaded area means std error.

semantically ungrounded.

5.2 How is the proposed method compared to prior work on zero-shot performance?

Next, we conducted experiments on prior UED methods with the use of a random environment generator (as described in Section 5.1), integration of those methods with a VLM-based generator (Variation + Visual + Think), and our proposed method.

We evaluated the following methods. (1) **DR (Domain Randomization)**: At the start of each episode, a new environment is sampled from the random generator. (2) **PLR** [18]:

doi:10.6342/NTU202501834

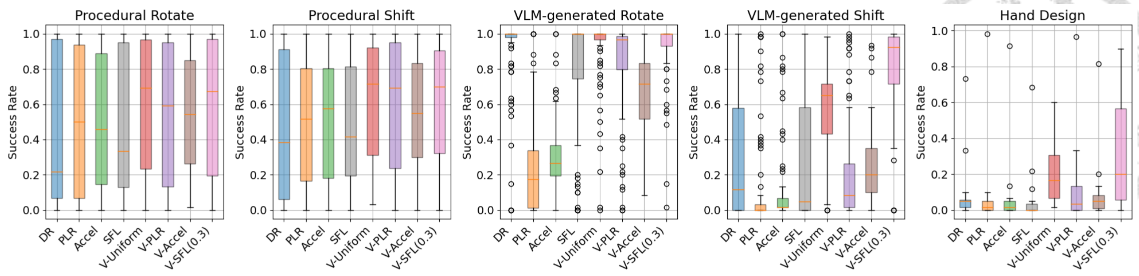
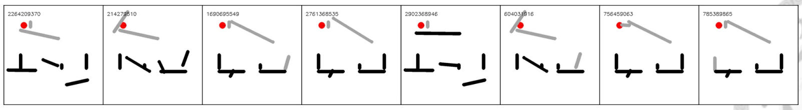


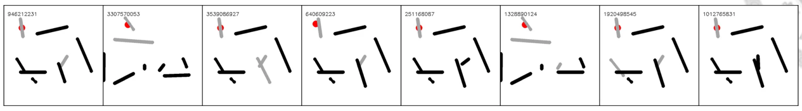
Figure 5.4: **Box plot of success rate.** VLM-based methods, the success rate distribution of V-Uniform/V-PLR/V-Accel/V-SFL uniformly improves on Procedural Rotate and Procedurall Shift.

Environments are prioritized based on positive value loss, using the robust variant **PLR**$^{\perp}$. (3) **ACCEL** [27]: An editor mutates environments by applying small offsets to the x and y coordinates of a randomly selected block or ball, and with 50% probability, flips the elimination property of the selected block. The mutation is applied to a source environment sampled from the random generator. (4) **SFL** [28]: In each learnability computation, we compute the learnability $p(1-p)$ of environments by first sampling 1000 environments and selecting the top 32 by learnability, following their recommended protocol. An additional 32 environments are sampled randomly, and the total 64 environments are used for training. Learnability is re-evaluated every 10 PPO iterations. (5) **V-Uniform**: We generate $N = 10000$ environments from a VLM-based generator and sample them uniformly. This can be viewed as an extended version of the setup in Section 5.1 with a larger environment pool. (6) **V-PLR**: This baseline uses the same environment pool as **V-Uniform** but applies the **PLR** prioritization strategy for sampling. (7) **V-ACCEL**: Similar in structure to **ACCEL**, but applied to VLM-generated environments instead of randomly generated ones. (8) **V-SFL (ours)**: The environments are generated as in **V-Uniform**, but sampled according to our method described in Section 4.3, with $\alpha = 0.3$ and learnability computed every $I = 10$ PPO iterations.

We evaluate the above methods through all test environment sets proposed in Sec-

doi:10.6342/NTU202501834


(a) Around 3M step, 2 edits


(b) Around 12M steps, 13 edits

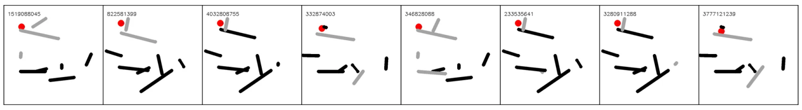

(c) Around 30M steps, 25 edits

Figure 5.5: **Environments generated by ACCEL.** ACCEL makes small edits to existing environments with high regret. However, visually, it drifts from the example drastically.

tion 4.4. We run the experiment for 30M environment steps. Following prior work, we use PPO [29] as the main RL algorithm. We note that PPO also behaves better compared to value-based methods due to the sparse-reward property of the task we studied. Figure 5.3 shows that V-SFL converges faster on VLM-generated Shift/Rotate while achieving a similar performance with V-Uniform on Procedural Shift/Rotate. Notably and somewhat surprisingly, we find that the zero-shot generalization result for DR sometimes even performs better than intricate methods such as PLR and ACCEL. This result does not match the improvements observed in those works. We believe that the root cause for this behavior is that the random generator could lead to unbounded complexity on the generated environments, and our use of a grounded VLM could counter this issue effectively. We also provide the box plot as lens into the distribution of success rate from different methods. Figure 5.4 implied that VLM-based methods uniformly improve the 25th and 75th percentiles on Procedural Rotate and Procedural Shift.

Analysis on ACCEL and V-ACCEL. In Figure 5.3, we notice that ACCEL/V-ACCEL behaves similar or worse to PLR/V-PLR and we do not observe gains of evolving complexity through mutations [27]. To understand this phenomenon, we look at the actual

doi:10.6342/NTU202501834

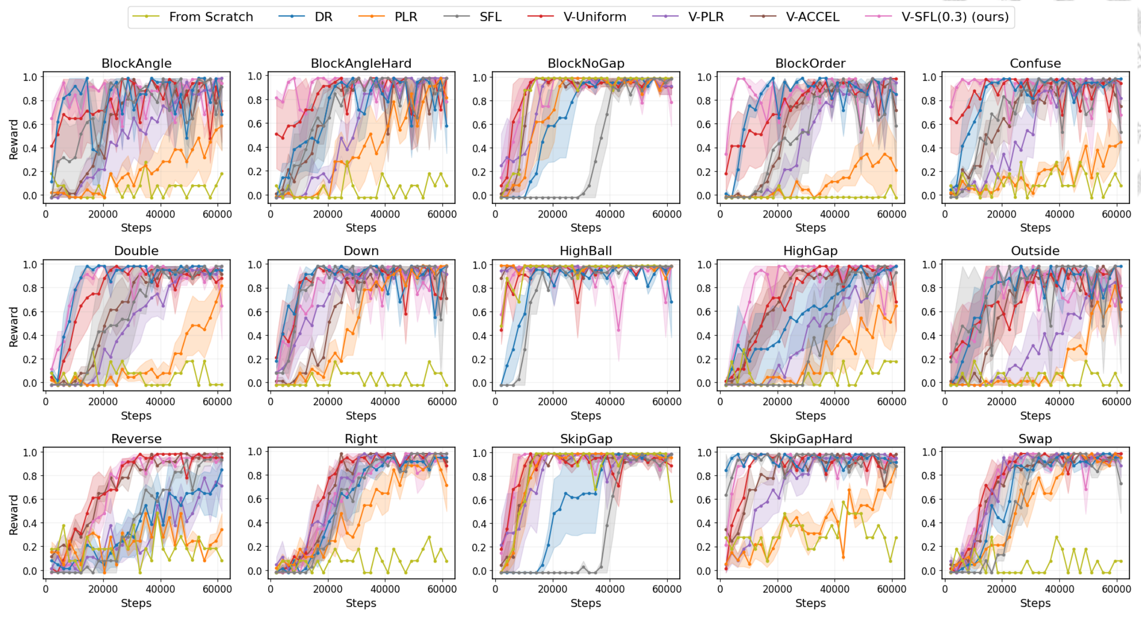


Figure 5.6: **Rapid adaptation on hand-designed test environments.** After training on a large number of environments, the model exhibits rapid adaptation on unseen hand-designed environments, even when the zero-shot success rate remains relatively low. This behavior uniformly holds.

environments after several mutations. As shown in Figure 5.5, we qualitatively identified that the generated environment will become very random and visually drifts from the example after several edits. This indicates an important limitation of ACCEL: the complexity brought by evolving the environments is uncontrollable, stressing the need to obtain a grounded environment generation.

5.3 How fast does the proposed method adapt to unseen environment(s)?

From the zero-shot generalization result, we find that the zero-shot success rate is still unsatisfactory for procedurally generated environments and hand-designed environments. Therefore, we consider a practical setup where it is affordable to finetune the trained policy model on an unseen environments(s).

doi:10.6342/NTU202501834

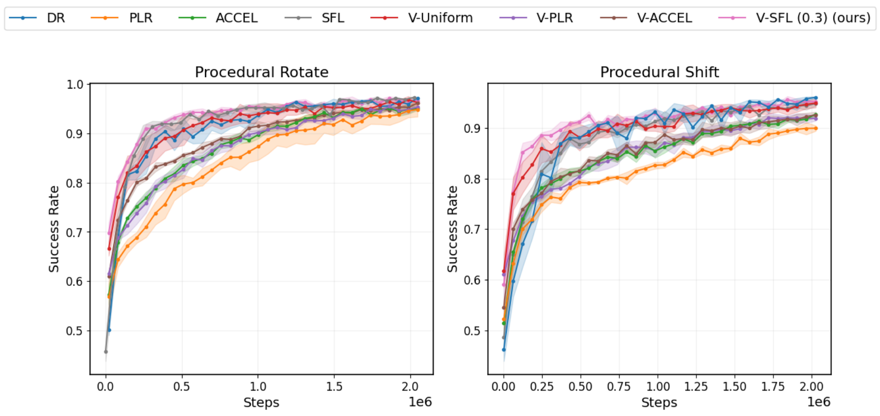


Figure 5.7: **Rapid multi-task finetuning on Procedural Rotate/Shift.** V-SFL outperforms baselines as it achieves more than 0.9 averaged success rate while using the smallest steps.

We evaluate how quickly the model could adapt to the unseen environments(s) in two setups. (1) **Single-task finetuning.** We picked the checkpoint around 30M step from each method and finetuned the model by each environment of the hand-designed environments individually. All model parameters are tunable. We also train a model from scratch for comparison. Figure 5.6 shows that after V-SFL is usually the first or second to get a success rate of more than 0.9, while other baselines usually struggle in certain environments. (2) **Multi-task finetuning.** We now consider the multi-task finetuning setup, where we finetune the checkpoint around 30M step of each method on Procedural Rotate and Procedural Shift. The finetuning takes 1.5M steps. Figure 5.7 represents that V-SFL converges drastically faster compared to other baselines, demonstrating that the initialization obtained by V-SFL is better.

The role of critic in finetuning. The regret approximation of PLR and ACCEL is based on value loss. Intuitively, they prioritize environments that are with large surprise on the expected value and actual value. Therefore, we investigate whether replacing the critic network would affect the progress of finetuning. We explore two settings: (1) **V-SFL with others' critic**: We replace the V-SFL critic with the critic of V-PLR, V-ACCEL, V-

doi:10.6342/NTU202501834

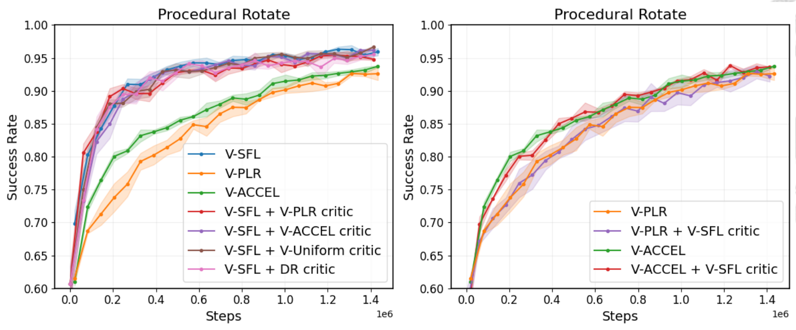


Figure 5.8: **Finetuning trend with replacing critic. Left**: V-SFL with others' critic. **Right**: V-PLR/V-ACCEL with V-SFL critic. Replacing critic results similarly, showing the important role of policy generalization.

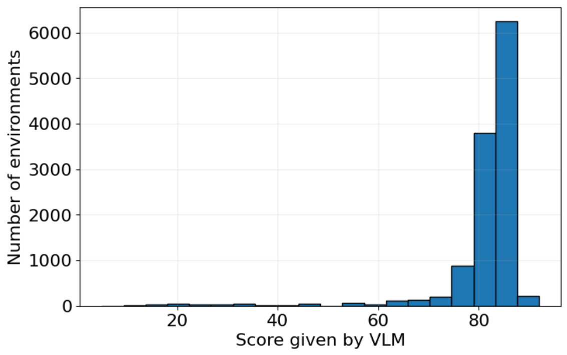


Figure 5.9: **Score given by a VLM.** The score VLM given to the generated environments almost falls between 80 and 90.

Uniform or DR. (2) **V-PLR/V-ACCEL with V-SFL critic**. Figure 5.8 shows the multi-task finetuning success rate trend. Replacing critic does not affect finetuning performance. We hypothesize that in sparse reward setup, the value function is noisy and unhelpful for achieve better generalization. Therefore, we think that regret approximation based on policy performance might be the key for future improvement.

5.4 Can VLM be used to score the environments and generate a curriculum based on the score?

We already see that VLM could be helpful in generating environments with grounded complexity that helps generalization. In previous experiments, we used a learnability-

doi:10.6342/NTU202501834

based curriculum to prioritize environments with more learning potential. Intuitively, could VLM also be used to prioritize the environments, given that it might contain knowledge about what are better environments for policy learning? Therefore, we asked another VLM to score all the generated environments. Similar to the generation prompt, we only changed its goal to produce a score, which the higher score should mean the environment is not too hard or too easy for learning. The score is between 0 and 100. The result is shown in Figure 5.9. Interestingly, the VLM could just produce scores in a very narrow region. We think that it indicates that physics-based control is a domain in which VLM rating is not useful because most environments are given a similar score.

doi:10.6342/NTU202501834

Chapter 6 Conclusion

In this work, we point out that, in our physics-based control benchmark, a grounded environment generator is crucial to achieve better generalization. We address the poor generalization issue in our physic-based control task by grounding a VLM-based environment generator with several important design choices, driven by the semantic groundedness and sample complexity groundedness metrics. Furthermore, we present V-SFL, a method that leverages VLM as the grounded environment generator and employs a learnability-based curriculum to achieve the best generalization result.

Although we have studied the generalization of physics-based control in a benchmark that offers a more isolated way for investigating the physical law generalization, how to connect the study into broader regime such as tasks with semantic understanding or even real-world robotic tasks are not yet explored. Furthermore, another limitation of our method is that it is required to pre-generate a big environment pool and so the agent learning might plateau at some point, while it is possible to be addressed by repeating the grounding process on the learned policy. An interesting future direction is how to post-train the VLM model and make it a teacher that could generate environments with high learnability directly. A starting point is to post-train the VLM by reinforcement learning and use regret as the reward. We hope our work could shed lights on how to make physics-based control generalize better.

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

References

[1] A. S. Azad, I. Gur, J. Emhoff, N. Alexis, A. Faust, P. Abbeel, and I. Stoica. Clutr: Curriculum learning via unsupervised task representation learning. In International Conference on Machine Learning, pages 1361–1395. PMLR, 2023.

[2] A. Bakhtin, L. van der Maaten, J. Johnson, L. Gustafson, and R. Girshick. Phyre: A new benchmark for physical reasoning. Advances in Neural Information Processing Systems, 32, 2019.

[3] D. M. Bear, E. Wang, D. Mrowca, F. J. Binder, H.-Y. F. Tung, R. Pramod, C. Holdaway, S. Tao, K. Smith, F.-Y. Sun, et al. Physion: Evaluating physical prediction from vision in humans and machines. arXiv preprint arXiv:2106.08261, 2021.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, jun 2013.

[5] M. Beukman, S. Coward, M. Matthews, M. Fellows, M. Jiang, M. Dennis, and J. Foerster. Refining minimax regret for unsupervised environment design. arXiv preprint arXiv:2402.12284, 2024.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakan-

doi:10.6342/NTU202501834

tan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[7] Z. Chen, K. Yi, Y. Li, M. Ding, A. Torralba, J. B. Tenenbaum, and C. Gan. Comphy: Compositional physical reasoning of objects and events from videos. arXiv preprint arXiv:2205.01089, 2022.

[8] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. Extreme parkour with legged robots. In 2024 IEEE International Conference on Robotics and Automation (ICRA), pages 11443–11450. IEEE, 2024.

[9] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. CoRR, abs/2306.13831, 2023.

[10] H. Chung, J. Lee, M. Kim, D. Kim, and S. Oh. Adversarial environment design via regret-guided diffusion models. arXiv preprint arXiv:2410.19715, 2024.

[11] M. Dennis, N. Jaques, E. Vinitsky, A. Bayen, S. Russell, A. Critch, and S. Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. Advances in neural information processing systems, 33:13049–13061, 2020.

[12] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. arXiv preprint arXiv:2004.07219, 2020.

[13] S. Garcin, J. Doran, S. Guo, C. G. Lucas, and S. V. Albrecht. Dred: Zero-shot transfer in reinforcement learning via data-regularised environment design. arXiv preprint arXiv:2402.03479, 2024.

doi:10.6342/NTU202501834

[14] Q. Garrido, N. Ballas, M. Assran, A. Bardes, L. Najman, M. Rabbat, E. Dupoux, and Y. LeCun. Intuitive physics understanding emerges from self-supervised pretraining on natural videos. arXiv preprint arXiv:2502.11831, 2025.

[15] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. Journal of Machine Learning Research, 23(274):1–18, 2022.

[16] S. James, Z. Ma, D. Rovick Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. IEEE Robotics and Automation Letters, 2020.

[17] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. arXiv preprint arXiv:2205.09991, 2022.

[18] M. Jiang, M. Dennis, J. Parker-Holder, J. Foerster, E. Grefenstette, and T. Rocktäschel. Replay-guided adversarial environment design. Advances in Neural Information Processing Systems, 34:1884–1897, 2021.

[19] M. Jiang, E. Grefenstette, and T. Rocktäschel. Prioritized level replay. In International Conference on Machine Learning, pages 4940–4950. PMLR, 2021.

[20] B. Kang, Y. Yue, R. Lu, Z. Lin, Y. Zhao, K. Wang, G. Huang, and J. Feng. How far is video generation from world model: A physical law perspective. arXiv preprint arXiv:2411.02385, 2024.

[21] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. Advances in neural information processing systems, 35:22199–22213, 2022.

doi:10.6342/NTU202501834

[22] S. Li, K. Wu, C. Zhang, and Y. Zhu. I-phyre: Interactive physical reasoning. arXiv preprint arXiv:2312.03009, 2023.

[23] W. Liang, S. Wang, H.-J. Wang, O. Bastani, D. Jayaraman, and Y. J. Ma. Environment curriculum generation via large language models. In 8th Annual Conference on Robot Learning, 2024.

[24] Z. Lin, Y.-F. Wu, S. Peri, B. Fu, J. Jiang, and S. Ahn. Improving generative imagination in object-centric world models. In International conference on machine learning, pages 6140–6149. PMLR, 2020.

[25] M. Matthews, M. Beukman, C. Lu, and J. Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. arXiv preprint arXiv:2410.23208, 2024.

[26] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. arXiv preprint arXiv:2111.05803, 2021.

[27] J. Parker-Holder, M. Jiang, M. Dennis, M. Samvelyan, J. Foerster, E. Grefenstette, and T. Rocktäschel. Evolving curricula with regret-based environment design. In International Conference on Machine Learning, pages 17473–17498. PMLR, 2022.

[28] A. Rutherford, M. Beukman, T. Willi, B. Lacerda, N. Hawes, and J. Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. arXiv preprint arXiv:2408.15099, 2024.

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

[30] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake. Do differentiable simulators

doi:10.6342/NTU202501834

give better policy gradients? In International Conference on Machine Learning, pages 20668–20696. PMLR, 2022.

[31] F.-Y. Sun, S. Harini, A. Yi, Y. Zhou, A. Zook, J. Tremblay, L. Cross, J. Wu, and N. Haber. Factorsim: Generative simulation via factorized representation. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024.

[32] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), pages 23–30. IEEE, 2017.

[33] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[34] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang. Gensim: Generating robotic simulation tasks via large language models. arXiv preprint arXiv:2310.01361, 2023.

[35] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. arXiv preprint arXiv:2311.01455, 2023.

[36] M. Wołczyk, B. Cupiał, M. Ostaszewski, M. Bortkiewicz, M. Zając, R. Pascanu, Ł. Kuciński, and P. Miłoś. Fine-tuning reinforcement learning models is secretly a forgetting mitigation problem. arXiv preprint arXiv:2402.02868, 2024.

doi:10.6342/NTU202501834

[37] Z. Wu, N. Dvornik, K. Greff, T. Kipf, and A. Garg. Slotformer: Unsupervised visual dynamics simulation with object-centric models. arXiv preprint arXiv:2210.05861, 2022.

[38] Z. Wu, J. Hu, W. Lu, I. Gilitschenski, and A. Garg. Slotdiffusion: Object-centric generative modeling with diffusion models. Advances in Neural Information Processing Systems, 36:50932–50958, 2023.

[39] K. Yi, C. Gan, Y. Li, P. Kohli, J. Wu, A. Torralba, and J. B. Tenenbaum. Clevrer: Collision events for video representation and reasoning. arXiv preprint arXiv:1910.01442, 2019.

[40] A. Yu, G. Yang, R. Choi, Y. Ravan, J. Leonard, and P. Isola. Learning visual parkour from generated images. In 8th Annual Conference on Robot Learning, 2024.

[41] A. Zook, F.-Y. Sun, J. Spjut, V. Blukis, S. Birchfield, and J. Tremblay. Grs: Generating robotic simulation tasks from real-world images. arXiv preprint arXiv:2410.15536, 2024.

doi:10.6342/NTU202501834

Appendix A — Environment generation prompts

To generate environments, we use the following prompt. We use gpt-4.1-2025-04-14 as the VLM. The temperature is 1. We note that the prompt is based on [22], and we simplify the configuration that omits dynamic and spring properties. Also, unlike the original configuration, we allow only a single red ball for simplicity. In principle, we try to update the prompt to contain most details that may affect the understanding of the game.

VLM as generator prompt

You are an expert in game design. We have a game of a simulated square-shaped 2D world of size 600*600 consisting of some objects. The goal of the game is to drop the ball into the abyss (very bottom of the world) by eliminating blocks that can be eliminated within 15.0 seconds.

Your goal: Generate a new game config by modifying the given game config as provided below.

Each game is represented by an object configuration array, in JSON format. The object configuration array is as follows (blocks and balls are all objects).

For blocks: The maximum number of blocks is 11. You can add or remove blocks. [[[x1, y1], [x2, y2]], eli] [x1, y1] means the left end point and [x2, y2] means the

doi:10.6342/NTU202501834

right end point. The block is always a rectangle. The block's height is always 20. eli: 0/1 means whether the corresponding object can be eliminated. 1 is eliminable and 0 is not eliminable. For balls: There is only one ball in the game. [x, y, radius] [x, y] means the center of the ball. 'radius' means the radius of the ball. The ball is always a circle. The ball's diameter is 2*radius. For example, if radius is 20, then the ball's diameter is 40.

In the world, there are the rules that objects follow: - Coordinates of objects are given by (x, y), where x is horizontal (0 to 600 represents left to right) and y is vertical (0 to 600 represents top to bottom). - During the simulation, only the ball can move. All the blocks are static. - If the agent do not want to eliminate a block, the agent can do nothing, which is also an action. - Note that if the block is horizontally placed, and the ball falls without horizontal force, it will just stay at the same position on the block.

The given game config is as follows:

Game name: hole_hard JSON config: "'json "'

Please follow the following format to respond: <think>Please use this area to think about your idea.</think> <description>Put the description here.</description> <new_game_config>Put the new game config here.</new_game_config>

Second, for evaluating the quality by another VLM, we use the following prompt. Most part are the same but the goal is different. To avoid creative result, we set temperature to 0.4.

doi:10.6342/NTU202501834

VLM as evaluator prompt

You are an expert in game design. We have a game of a simulated square-shaped 2D world of size 600*600 consisting of some objects. The goal of the game is to drop the ball into the abyss (very bottom of the world) by eliminating blocks that can be eliminated within 15.0 seconds.

Your goal: We have a game config. Please evaluate the given game config in terms of how much spatial reasoning learning opportunity it provides for a RL learning agent, so the agent can perform well in the game after training. The best game should be neither too easy nor too hard.

Each game is represented by an object configuration array, in JSON format. The object configuration array is as follows (blocks and balls are all objects).

For blocks: The maximum number of blocks is 11. You can add or remove blocks. [[[x1, y1], [x2, y2]], eli] [x1, y1] means the left end point and [x2, y2] means the right end point. The block is always a rectangle. The block's height is always 20. eli: 0/1 means whether the corresponding object can be eliminated. 1 is eliminable and 0 is not eliminable. For balls: There is only one ball in the game. [x, y, radius] [x, y] means the center of the ball. 'radius' means the radius of the ball. The ball is always a circle. The ball's diameter is 2*radius. For example, if radius is 20, then the ball's diameter is 40.

In the world, there are the rules that objects follow: - Coordinates of objects are given by (x, y), where x is horizontal (0 to 600 represents left to right) and y is vertical (0 to 600 represents top to bottom). - During the simulation, only the ball can move. All the blocks are static. - If the agent do not want to eliminate a block, the agent can do nothing, which is also an action. - Note that if the block is horizontally

doi:10.6342/NTU202501834

placed, and the ball falls without horizontal force, it will just stay at the same position on the block.

The given game config is as follows:

JSON config: “‘json “‘

Please follow the following format to respond: <think>Please use this area to think about it.</think> <evaluation>Put your evaluation after thinking here.</evaluation> <score>Put your score here. should be just a number between 1 and 100 (the bigger the better).</score>

doi:10.6342/NTU202501834

Appendix B — Details of procedurally generated environments

In the main text, we mentioned that there are two procedurally generated environments: shift and rotate. In particular, environments of rotate also include shift.

Throughout the whole paper, we use a sample game as template to reduce the possible space. The JSON of the sample game is:

doi:10.6342/NTU202501834

Sample game config

```
{
    "block": [
        [[100.0, 150.0], [330.0, 200.0]],
        [[160.0, 100.0], [160.0, 130.0]],
        [[100.0, 400.0], [250.0, 400.0]],
        [[350.0, 400.0], [500.0, 400.0]],
        [[500.0, 300.0], [500.0, 380.0]],
        [[250.0, 360.0], [250.0, 380.0]],
        [[350.0, 360.0], [350.0, 380.0]],
        [[100.0, 300.0], [100.0, 380.0]],
    ],
    "ball": [[120.0, 120.0, 20.0]],
    "eli": [1, 1, 0, 0, 0, 0, 0, 0, 0],
    "dynamic": [0, 0, 0, 0, 0, 0, 0, 0, 1],
}
```

Here, we explain how those environments are generated.

For shift environments, for the first and second block, we sample offset_x in $[-100, 400]$ and offset_y in $[-100, 300]$ and add offset_x and offset_y to the value of x and y for the first two blocks. For other blocks, we sample offset_other_x in $[-100, 100]$ and offset_other_y in $[-100+\text{offset\_y}, \min(100+\text{offset\_y}, 180)]$ and add to the x, y position of the remaining blocks. Moreover, the ball's x position is added a value sampled from $[-30, 200]$ and y is added a value sampled from $[-100, 0]$.

doi:10.6342/NTU202501834

For rotate environments, we follow the shift environments' method to shift blocks but do not shift the ball. Instead, the ball is shifted by offset_x and offset_y from the first two blocks. Then, we sample a value of angle in $[-\pi/4, \pi/4]$ and rotate the first two blocks as well as the ball with respect to the first block's center by angle degree.

Finally, for all procedurally generated environments, if the ball's x or y is less than 20 or more than 580, the environment is excluded from the result.

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

Appendix C — Experiment details

Following [22], we use the same setting in the action space and the observation space. In particular, the action space is 7, and the observation space is symbolic space with size 12 x 9 + 7 x 2 = 122, where the former refers to block states and the latter refers to action states. For block states, 12 is the upper limit of the number of blocks/balls and for each block/ball it uses 9 dimensions to describe it. We used only the elimination and the dynamic property in our experiment, and other properties were not used. For action states, 7 is the limit of possible actions, and for each action, the 2 dimensions are used to indicate which position it would click if the action is used.

In our experiments, we use the same model architecture. The policy model contains encoding layers and 3-layer MLPs. The input is the symbolic space with 122 dimensions. For a block state, the input dimension is 9, and it is encoded by a hidden layer of 32 and then a layer of 16. For an action state, it is encoded by a hidden layer of 8 and then by a layer of 16. The logits from all block encoders and action encoders are concatenated in a permutation-invariant way to avoid the sensitivity of block order. Finally, the hidden dimension of the MLP is 256 and 256. The output of the policy is a probability distribution of actions. The critic model is only different from the policy model in the output, which the critic model outputs a scalar, and the policy model outputs a vector which the size is the action space. The policy model does not share any parameter with the critic model.

doi:10.6342/NTU202501834

We provide all hyperparameters in Table C.1. Our implementation is based on [18] at https://github.com/facebookresearch/dcd/. For the first experiment, our PPO implementation is based on CleanRL [15].

doi:10.6342/NTU202501834

Parameter	Value
PPO	
Number of workers	64
Entropy coefficient	0.01
Reward scale	0.001
Number of steps	256
Number of minibatches	4
Update epochs	30
Advantage normalization	No
Value loss clipping	No
Value loss function	MSE
Action repeat	4
Frame skip	6
Frame rate	60 fps
Anneal LR	Yes
Discounted factor	0.99
GAE lambda	0.995
Clip coefficient	0.2
Max gradient norm	0.5
Target KL	0.3
PLR, V-PLR	
Level replay strategy	positive value loss
Level replay probability	0.5
ρ	0.5
Buffer size	1000
Staleness	0.5
ACCEL, V-ACCEL	
Probability of edit	1.0
Num of edits	1
SFL	
Update per PPO iteration	10
N	1000
NL	64
K	100
Sampling	Uniform from Top K
Staleness	0.5
V-SFL	
Update per PPO iteration I	10
Buffer size	1000
Sampling	Weighted by Score
Staleness	0.1

Table C.1: Hyperparameters used in our experiments.

doi:10.6342/NTU202501834

doi:10.6342/NTU202501834

Appendix D — On the impact of different α

We conducted experiments on how the value of α affects the generalization result. As shown in Figure D.1, we find that α = 0.1 and 0.3 generally works better in VLM-generated environments. We used $N = 10000$ in this experiment.

doi:10.6342/NTU202501834

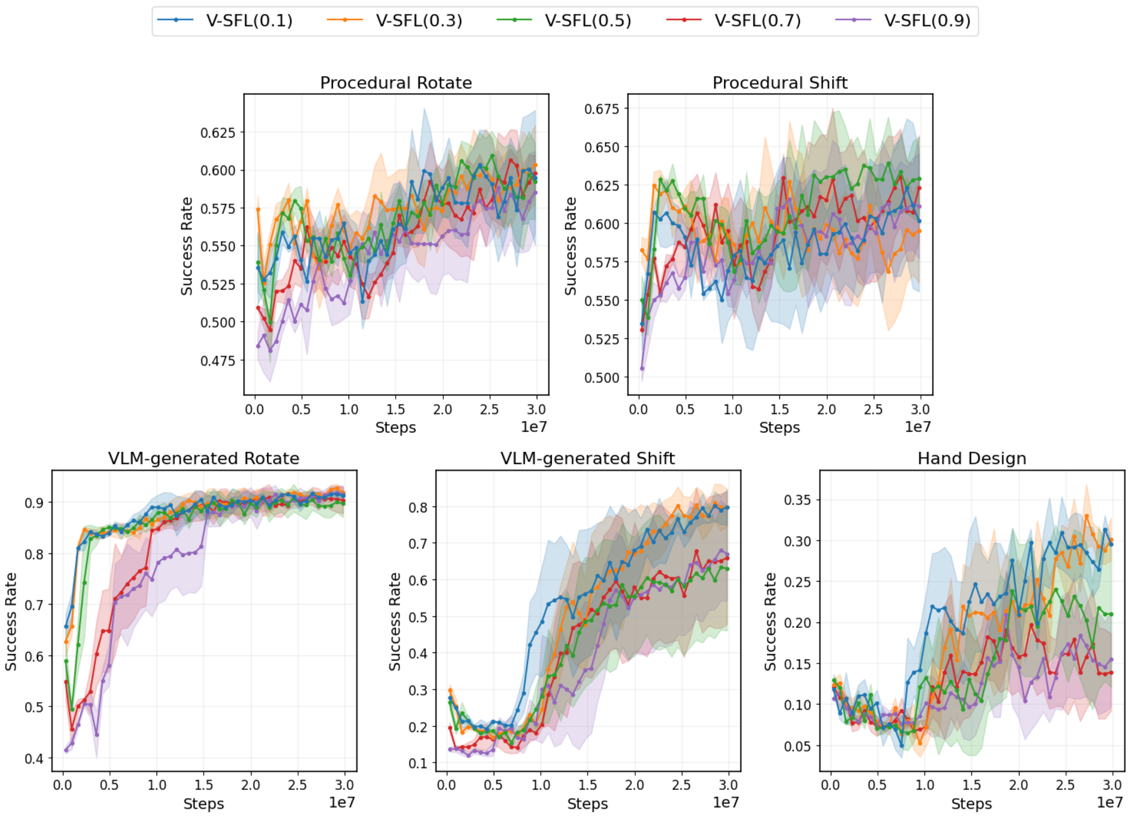


Figure D.1: **Impact of different α.**

doi:10.6342/NTU202501834

Appendix E — Hand-designed test environments

Future E.2 provides the visualization of the hand-designed test environments. We design those environments by trying to create some adversarial environments which we think the model could be confused about.

doi:10.6342/NTU202501834

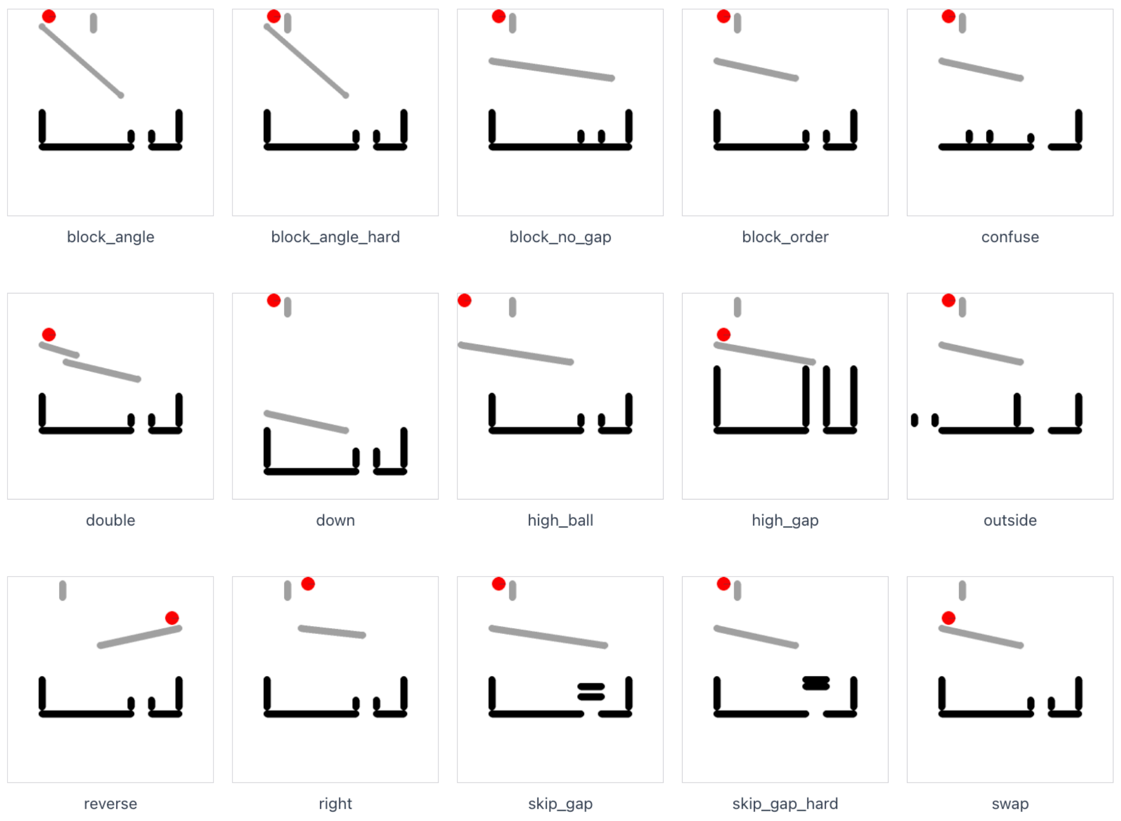


Figure E.2: **Hand designed test environments.** A total of 15 hand-designed test environments are used for evaluation.

doi:10.6342/NTU202501834