Improving Robot Policies Through Failure-Driven Scene Generation

Rohan Yelandur, Ayaan Sunesara, Druhinh Roy, Vedant Sangani

***Abstract*—Creating generalist robot manipulation policies is difficult as they often fail due to rare or unseen environmental factors. Current approaches use heuristic approaches to improve the model's capabilities by first identifying failure modes and then effectively training on new scenes. We propose a unified framework for failure-driven scene generation that enables a robot to identify its own weaknesses and retrain itself to improve robustness. Our method integrates a failure-analysis module that ranks error types, a scene generator that produces tailored scenes to those failures, and finally an adaptive training loop that fine-tunes the policy using those scenes with a meaningful training distribution. Project repository: https://github.com/Rohan-Yelandur/robosuite-scene-generation.**

I. Introduction

The achievement of generalist robot manipulation remains a fundamental challenge in robotics. Although advances in imitation learning allow robots to master skills in controlled environments, these policies can be brittle in the real world. Seemingly minor changes in lighting, object placement, or textures can cause failures in policies that excel during training.

Common approaches like domain randomization or exhaustive data augmentation face significant efficiency problems. Training in random variations can be redundant and covering every possible environmental configuration is nearly impossible. This is why robots trained in single environments struggle to generalize well. Indiscriminate scaling up of the data fails to guarantee coverage of the specific edge cases that cause policy collapse.

We believe that robustness requires training in the right environments rather than all possible ones. It is more efficient to identify and target the specific factors that cause failure.

We propose a unified framework for failure-driven scene generation. This closed-loop system enables a robot to self-diagnose weaknesses and autonomously curate its training curriculum. Unlike heuristic approaches that rely on manual failure identification, our method automates the discovery of vulnerabilities.

Our framework operates through these three integrated stages:

1) **Failure Analysis:** A module that evaluates policy performance to rank and categorize error types.
2) **Targeted Scene Generation:** A generative mechanism that produces novel environmental configurations tailored to provoke identified failure modes.
3) **Selective Fine-tuning:** A fine-tuning loop that updates the policy using this meaningful distribution of adversarial scenes.

By shifting from using broad data to targeted training, we will show that robotic policies can improve their robustness with improved data efficiency. This approach allows the agent to iteratively harden itself against rare or unseen factors that would otherwise fail during deployment.

II. Related Work

Training robust robotic policies requires exposure to diverse environments that often challenge agent generalization. Prior work has explored things like domain randomization, adversarial training, and automated scene generation to address policy brittleness, however these approaches either waste training capacity on trivial variations or generate unsolvable tasks.

A very significant contribution to this space came from the paper Emergent Complexity and Zero-Shot Transfer via Unsupervised Environment Design [3], which introduced Unsupervised Environment Design (UED), framing environment generation as an adversarial learning problem and focusing on creating high regret for the learning agent rather than relying on manually specified environment distribution. The proposed PAIRED framework balances task difficulty by using a protagonist–antagonist setup, enabling the generation of complex but solvable environments and improving zero-shot transfer. This work addresses key limitations of prior approaches, such as domain randomization, which often wastes training capacity on trivial data, and naive minimax adversarial training, which can generate impossible or degenerate tasks, both which harm the success of the policy. A Related adversarial robustness solution similarly generates worst-case environments, but without explicit curriculum balancing or failure targeting [9].

Recent work has further explored procedural and learned scene generation to increase the diversity of training environments. ClutterGen [4] rearranges object configurations to produce cluttered manipulation scenes, while DynScene [5] scales dynamic scene generation using text prompts to specify environments and actions. Steerable Scene Generation [8] introduces diffusion-based models that enable post-training control and inference-time search over generated scenes. Large-scale environment and task generation using language models has also been investigated [13, 6], where LLMs generate task specifications and environment parameters to increase task diversity. Similarly, domain-randomized benchmarks such as RoboTwin 2.0 emphasize the scale of simulated data generation [2]. However, these approaches generally lack mechanisms to adapt scene generation based on observed policy failures.

As we continue to understand this field, we find that understanding and exploiting the policy failures is key to improving robotic robustness. RoboFail [10] and related work on failure diagnosis [11] analyze failure cases to identify policy weaknesses, while runtime monitoring approaches study consistency and progress violations in generative policies [1]. These methods primarily focus on post-hoc analysis or monitoring rather than actively generating new environments to induce failures. Our goal has been aligned with this through the failure-centric perspectives, but differs in that failures are used online to guide environment generation, rather than solely for diagnostics. But by further understanding this, we can learn to integrate failure discovery with adversarial scene generation, and further boost our policy.

All experiments build on established robotic simulation infrastructure such as robosuite [14], which provides standardized manipulation environments and benchmarks. For policy optimization, we rely on Proximal Policy Optimization (PPO) [12], a widely used on-policy reinforcement learning algorithm known for stability and scalability in continuous control tasks.

III. Methodology

Our unified framework aims to improve robotic policy robustness by autonomously identifying weaknesses and generating targeted training data. The system operates as a closed-loop pipeline consisting of three core components: Failure Analysis, Targeted Scene Generation, and Selective Retraining.

In this section, we detail how we utilized the **RoboMD** framework [11] to diagnose our policies trained on the **Robomimic** dataset [7]. Specifically, we employed its discrete-continuous pipeline to automatically discover and quantify failure modes by exploring variations in environment conditions.

A. Failure Analysis with RoboMD

The goal of this module is to automatically discover and quantify the failure modes of our pre-trained robot manipulation policy. Rather than randomly guessing environmental parameters, we use an automated agent to systematically explore variations in the environment to find exactly where the policy breaks.

1) Discrete Stage: We begin with a predefined set of discrete environment variations. In our experimental setup, these included specific changes such as object properties, lighting colors, and table textures. A deep Reinforcement Learning (RL) agent treats these variations as a set of distinct ”actions” it can take.

The agent explores these actions to identify which specific conditions cause the robot to fail. This stage provides us with a starting set of known failure cases, but it is limited by the fixed list we defined beforehand.

2) Continuous Embedding Stage: To discover failures beyond our predefined list, we move to a continuous stage. We train a Vision-Language Model (VLM) that produces continuous embeddings. These embeddings represent different environment variations and their impact on the robot's success or failure. The embedding maps different changes into a continuous vector space where semantically similar conditions are located near each other.

Instead of selecting from discrete candidates, the RL agent now outputs continuous vectors within this embedding space. These vectors represent novel, semantically meaningful environment variations that go beyond the original discrete list.

3) How Continuous Exploration Works: In this phase, the embedding space effectively becomes the agent's action space. The RL policy observes the current state and outputs a position in this continuous embedding space. Our system then maps this embedding back to a meaningful environment perturbation that the robot allows us to execute.

We can then test whether the robot policy fails under that specific perturbation. Based on the result, the RL policy updates itself, learning to navigate toward regions of the embedding space that have a higher likelihood of causing failure. This allows the agent to explore the failure space by navigating this continuous embedding, enabling the interpolation and discovery of new failure conditions that were not explicitly programmed.

B. Targeted Scene Generation

The output of the failure analysis is a list of specific configurations that we know are likely to cause failure. We will use these settings to generate a new dataset of challenging scenes. This ensures that the new training data focuses specifically on the robot's current weaknesses rather than just giving it more random data that it already knows how to handle.

C. Selective Fine-tuning

We finish off by fine-tuning the robot on this new dataset. We would use an adaptive training loop where the policy is updated using the difficult scenes generated in the previous step. This allows the robot to practice specifically on the things it got wrong, which improves its robustness much more efficiently than standard training methods.

D. Fine-tuning Using PAIRED Algorithm

Since RoboMD conceptually is trained to recognize and rank failures within a policy given a list of environment modifications, it is possible to use a modified version of RoboMD as an adversary, based on its initial understanding of the failure distribution of the action space, to generate environments for the manipulation policy the protagonist to train, modeling the PAIRED algorithm [3]. The antagonist in this case is an older version of the checkpoint, and this proves to be a possible limitation due to the lack of experimentation with other choices of the antagonist. However, the benefit still stands, instead of RoboMD providing a static failure analysis of the policy that can be used to select environments to train on, an adversary reacts to the protagonist's progress and continues to find environment modifications that challenge the protagonist.

E. Designing an Adversary

To make an adversary that properly chooses feasible but challenging environments to the protagonist, an agent representing RoboMD's PPO agent's structure is used with a reward function that maximizes the reward of the antagonist subtracted by the reward of the protagonist for choosing a specific action. Additionally, the agent is penalized for repeating actions or choosing actions that result in near-zero success rates for both the antagonist and protagonist, and the adversary has a limit of how many actions it can apply on the environment.

IV. Experiments

We evaluate our proposed failure-driven scene generation framework on the "Lift" task from the Robosuite benchmark. Our experiments aim to answer the following questions:

1) Can the framework effectively identify and isolate specific failure modes in a trained robot policy?
2) How can we reduce the failure rates of the highest ranking failure modes with failure-guided fine-tuning in order to create a more uniform failure distribution across different failure modes?

A. Experimental Setup

a) Environment: We focus on the Lift task, where a robot arm must pick up a red cube from a white table until it reaches a certain height. The environment observations consist of 84×84 RGB images and the robot's joint and position states.

b) Base Policy: We utilize a standard image-based policy trained using behavior cloning via the Robomimic dataset. This policy performs at a 100% success rate in the original environment but has not been exposed to significant environment changes.

c) Environment Modification Space: To stress-test the policy, we defined a structured space of actions representing modifications to the environment taken from [11] categorized into:

- **Visual Perturbations:** Changes to lighting color (Red, Green, Blue, Gray), robot color, and object colors (Cylinder, Table).
- **Physical Perturbations:** Variations in the dimensions of the manipulation objects (Cylinder radius/height, Box length/width).

TABLE I: Action Space for Lift Task

ID	Action Description
0	Change the cube color to red
1	Change the cube color to green
2	Change the cube color to blue
3	Change the cube color to gray
4	Change the table color to green
5	Change the table color to blue
6	Change the table color to red
7	Change the table color to gray
8	Resize the table to dimensions (0.8, 0.2, 0.025)
9	Resize the table to dimensions (0.2, 0.8, 0.025)
10	Resize the cube to dimensions (0.04, 0.04, 0.04)
11	Resize the cube to dimensions (0.01, 0.01, 0.01)
12	Resize the cube to dimensions (0.04, 0.01, 0.01)
13	Change the robot color to red
14	Change the robot color to green
15	Change the robot color to cyan
16	Change the robot color to gray
17	Change the lighting color to red
18	Change the lighting color to green
19	Change the lighting color to blue
20	Change the lighting color to gray

d) Failure Discovery: We trained an adversarial agent using PPO to interact with the environment by selecting these perturbations as "latent actions" at the start of each episode. The adversary is rewarded for causing the base policy to fail, effectively seeking out the most challenging configurations.

B. Failure Mode Identification

We trained RoboMD's [11] PPO agent for 3000 episodes to apply environment modifications in steps in order to analyze failure modes within the Robomimic BC policy. The overall failure rate was 60.3% (1,190 failures), indicating that the PPO agent successfully found configurations that break the policy.

The framework successfully ranked the 20 distinct perturbation types by their likelihood to induce failure, revealing a clear hierarchy of vulnerabilities as shown in 1

C. Efficiency Analysis

These results demonstrate the inefficiency of standard Domain Randomization. A uniform DR approach would sample all perturbation types with equal probability. Our analysis shows that approximately half of the defined parameter space (e.g., robot colors, lighting, shrinking the cube) is already well-handled by the policy (0–20% failure).

Uniformly sampling these "safe" scenes would waste significant training compute. In contrast, our framework identifies that three specific modes (Increasing the cube size, Changing the Table Color to Red) account for the vast majority of failures. A targeted retraining curriculum focusing on these specific generated scenes would theoretically improve robustness with significantly higher sample efficiency than broad domain randomization.

D. Fine-tuning with RoboMD as an Adversary

Using the PAIRED [3] algorithm discussed earlier with a modified version of RoboMD serving as the adversary, the effectiveness of using failure analysis while fine-tuning

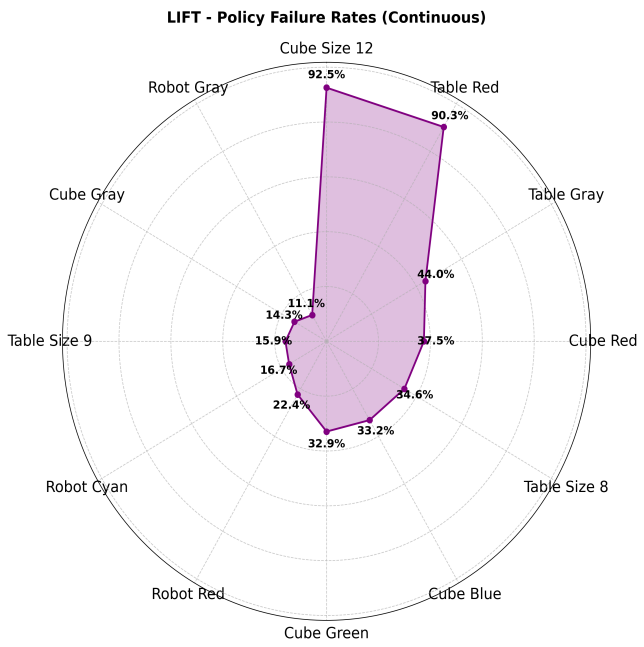


Fig. 1: Top 12 Actions Ranked by their Likelihood to Cause Failure Using RoboMD

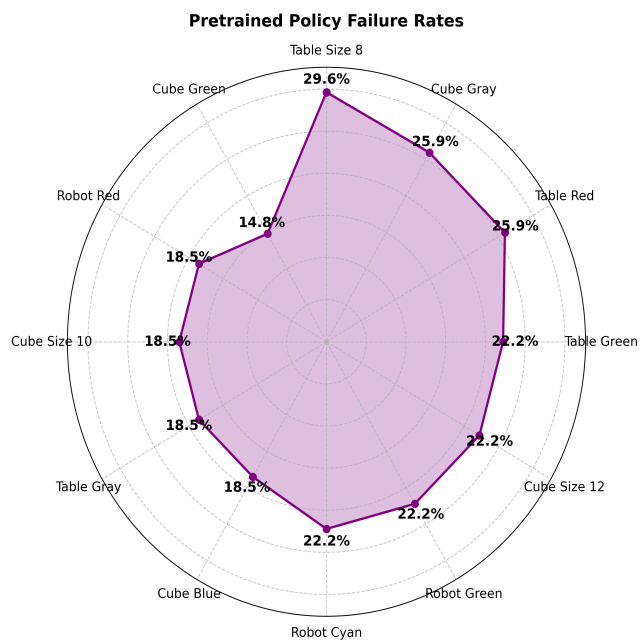


Fig. 2: Top Failure Rates of Applying Singular Actions for Pretrained Robomimic Policy

a policy on a set of environment modifications not originally in the dataset of a pretrained policy.

Once again, we utilize a standard image-based policy trained using behavior cloning via the Robomimic dataset for the Lift task. The environment modifications are the same 20 in Table I.

E. Performance Before Fine-tuning

The testing method used to test the performance of the policy before, during, or after fine-tuning involves applying singular actions from Table I over 27 trials and measuring success or failure. This is inherently flawed and a limitation to our paper due to the fact that multiple actions may be applied until a failure is reached by our adversary, but it may give untrustworthy insight of the impacts of fine-tuning.

Figure 2 represents the performance of the policy using this testing methods before fine-tuning. Note that parts of the rankings of the failure likelihoods do not match the rankings of the failure likelihoods found by RoboMD since the RoboMD agent was allowed to apply multiple actions until a failure was reached.

F. Fine-tuning Process

With the PAIRED algorithm, the BC model was trained for 1.2 million timesteps with the adversary receiving PPO updates at a ratio of 1:3 compared to the protagonist's PPO update rate in order to ensure that the protagonist has enough time to learn the current set of challenging environments presented by the adversary before the adversary updates. The maximum number of actions that could be applied to the environment was 3.

A limitation faced by our methodology was the hyperparameters of the PPO model of the adversary which was heavily encouraged to explore along with the weights of the reward function. This led to the adversary creating extremely challenging environment configurations which presumably decreased the effectiveness of the fine-tuning since the protagonist and adversary's mean reward both decreased to essentially 0 during the middle of fine-tuning as shown in Figure 3a and Figure 3b.

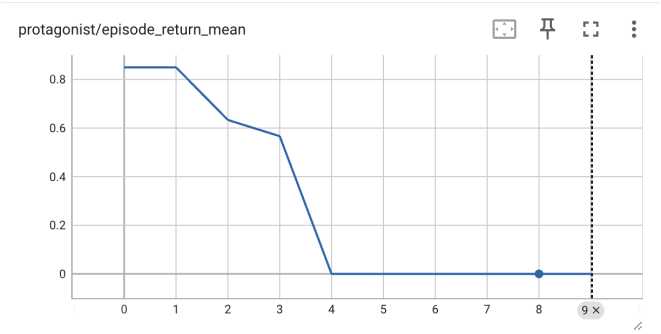
G. Results of Fine-tuning

Although there were many limitations in the training and testing methods, the results of fine-tuning shows improvement especially in reducing both the overall failure rate of the policy and making the failure distribution across the action space more uniform as shown in Figure 4.

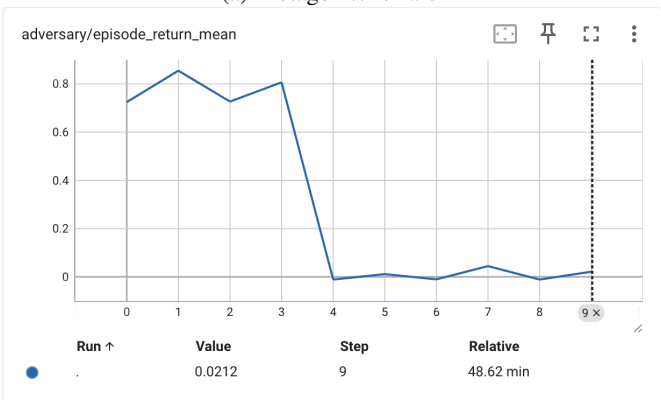
To measure whether failure-guided fine-tuning improves upon domain randomization, it would be necessary to train a policy with environments generated by domain randomization for the same number of episodes as the policy trained with failure-guided fine-tuning and compare the failure distributions of both.

V. Conclusion

In this paper, we introduced a unified framework for failure-driven scene generation to improve generalization in robot manipulation. Unlike standard domain randomization that uses blind sampling, our method targets the specific configurations that cause policy failure. Experiments on the Robosuite "Square" task revealed clear patterns of vulnerability. These results confirm that standard imitation learning policies often rely on superficial features, such as lighting color, rather



(a) Protagonist reward



(b) Adversary reward

Fig. 3: Mean rewards during fine-tuning

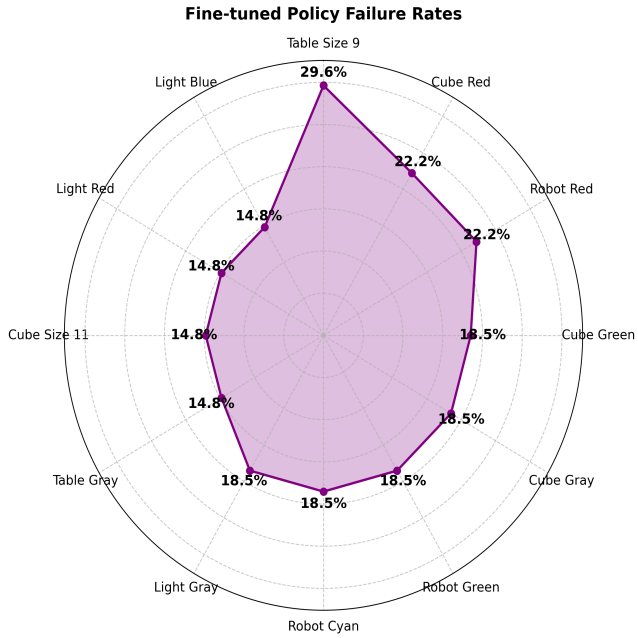


Fig. 4: Top Failure Rates of Applying Singular Actions for Fine-tuned Robomimic Policy

than robust physical understanding. While the base policy handled movement changes well, it failed consistently with specific lighting and geometric shifts. This validates the need for targeted training curricula rather than uniform random sampling.

Our current framework has limitations. It operates within a fixed set of possible changes and lacks a curriculum that adjusts difficulty based on learning progress. Future research will address this by automatically classifying and adding new action types based on failures observed in the real world. We also plan to use a dedicated antagonist agent to optimize scenes for learning potential rather than just difficulty. Finally, we will compare our failure-guided fine-tuning against training on all possible scene combinations to measure specific efficiency gains.

References

[1] Christopher Agia, Rohan Sinha, Jingyun Yang, Zi-ang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. *arXiv preprint arXiv:2410.04640*, 2024.

[2] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Qiwei Liang, Zixuan Li, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[3] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems*, volume 33, pages 13049–13061, 2020.

[4] Yinsen Jia and Boyuan Chen. Cluttergen: A cluttered scene generator for robot learning. In *8th Annual Conference on Robot Learning*, 2024.

[5] Sangmin Lee, Sungyong Park, and Heewon Kim. Dynscene: Scalable generation of dynamic robotic manipulation scenes for embodied ai. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 12166–12175, 2025.

[6] William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models. *arXiv preprint arXiv:2411.01775*, 2024.

[7] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation, 2021. URL https://arxiv.org/abs/2108.03298.

[8] Nicholas Pfaff, Hongkai Dai, Sergey Zakharov, Shun Iwase, and Russ Tedrake. Steerable scene generation with post training and inference-time search. *arXiv preprint arXiv:2505.04831*, 2025.

[9] Allen Z Ren and Anirudha Majumdar. Distributionally robust policy learning via adversarial environment generation. *IEEE Robotics and Automation Letters*, 7(2): 1379–1386, 2022.

[10] Som Sagar and Ransalu Senanayake. Robofail: Analyzing failures in robot learning policies. *arXiv e-prints*, pages arXiv–2412, 2024.

[11] Som Sagar, Jiafei Duan, Sreevishakh Vasudevan, Yifan Zhou, Heni Ben Amor, Dieter Fox, and Ransalu Senanayake. From mystery to mastery: Failure diagnosis for improving manipulation policies. *arXiv preprint arXiv:2412.02818*, 2024.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017. URL https://arxiv.org/abs/1707.06347.

[13] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023.

[14] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.