Position: Automatic Environment Shaping is the Next Frontier in RL

Younghyo Park [*1] **Gabriel B. Margolis** [*1] **Pulkit Agrawal** [1]

Abstract

Many roboticists dream of presenting a robot with a task in the evening and returning the next morning to find the robot capable of solving the task. What is preventing us from achieving this? Sim-to-real reinforcement learning (RL) has achieved impressive performance on challenging robotics tasks, but requires substantial human effort to set up the task in a way that is amenable to RL. It's our position that algorithmic improvements in policy optimization and other ideas should be guided towards resolving the primary bottleneck of shaping the training environment, i.e., designing observations, actions, rewards and simulation dynamics. Most practitioners don't tune the RL algorithm, but other environment parameters to obtain a desirable controller. We posit that scaling RL to diverse robotic tasks will only be achieved if the community focuses on automating environment shaping procedures.

1. Introduction

The advent of foundation models for speech, vision, and language processing has revolutionized Artificial Intelligence. It is widely believed that robotics is the next frontier, and the race to develop a foundation model for robotics is ongoing. The critical challenge in this pursuit is the limited availability of robotic data: trajectories of observations and robot actions. This starkly contrasts with computer vision and natural language processing, where large amounts of readily available internet data can be used. One way to gather robotic data is to deploy robots for many tasks in diverse environments. However, such deployment is only feasible if robots create value, i.e., successfully solve the task often enough. The result is a chicken-and-the-egg situation – robots need to be useful to be deployed, but for them to be useful requires collecting enough data to train a controller that creates value. The real question, therefore, is bootstrapping data collection.

A natural way to collect data is by teleoperating a robot to perform diverse tasks. However, this paradigm is challenging to scale as the human effort grows linearly with the amount of data that needs to be gathered. To ease data collection, recent works have made teleoperation more capable and easier (Zhao et al., 2023; Fu et al., 2024), but it doesn't change the linear scaling. At some point in the future, it is plausible that we will have enough data to train a large model that will reduce the number of demonstrations required to learn a new task and make the human effort sub-linear. However, we are far from that point. The primary reason is that policies obtained via supervised learning on a small dataset of demonstrations (i.e., learning from demonstration) have limited robustness and generalization and, therefore, cannot be used to collect data autonomously in more diverse settings.

In theory, given a reward function or by inferring a reward function from the demonstrations, autonomous data collection is possible using reinforcement learning (RL). It has been hard to realize this promise of RL because training in the real world often requires babysitting the robot to ensure safe operation, ensuring the reward function is not hacked, performing resets, and the data inefficiency of RL algorithms means they run for a long time before useful behaviors are discovered. Real-world RL training is an active area of research (Luo et al., 2024), and recent work has shown the plausibility of learning locomotion behaviors from real-world RL training (Wu et al., 2023; Lei et al., 2023). However, real-world RL is yet to achieve state-of-the-art robotic controllers, which means the data it generates is sub-optimal.

A related line of work bypasses the difficulty of training in the real world by training with RL in simulation and then successfully deploying policies in reality (i.e., sim-to-real RL). Such training has achieved state-of-the-art behaviors across many robot morphologies and complex tasks involving legged locomotion (Miki et al., 2022; Ji et al., 2023; Hoeller et al., 2023; Zhuang et al., 2023; Cheng et al., 2023; Jenelten et al., 2024), dexterous manipulation (Andrychow-

[*]Equal contribution [1]Improbable AI Lab, Massachusetts Institute of Technology, Cambridge, MA, USA. Correspondence to: Younghyo Park <younghyo@mit.edu>, Gabriel B. Margolis <gmargo@mit.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

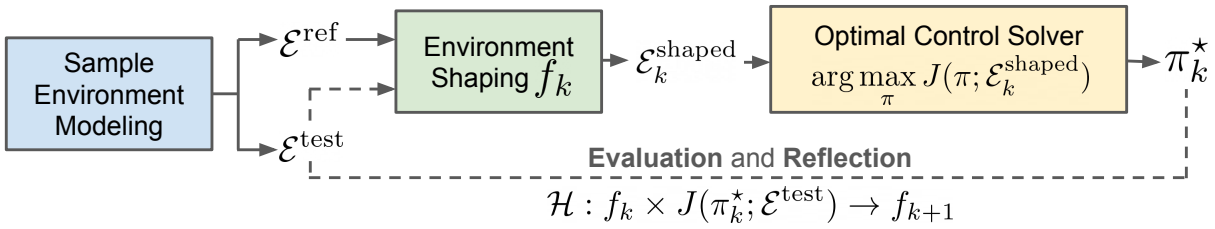

Figure 1. Flowchart of a typical behavior generation pipeline using reinforcement learning with simulation, illustrating four distinct subtasks of sample environment modeling, environment shaping, RL training, and outer feedback loop with behavior evaluation and reflection. We highlight the manual, task-driven **environment shaping** as a key, yet often overlooked, bottleneck in generalizing the success of RL. We thus advocate for automating the environment shaping process to broaden RL's applicability.

icz et al., 2020; Chen et al., 2023; Handa et al., 2023; Yin et al., 2023), drone racing (Song et al., 2023; Kaufmann et al., 2023) and others. A single general-purpose RL algorithm, PPO (Schulman et al., 2017), has powered these successes. Consequently, one might believe that a recipe for scaling to diverse tasks and collecting a sizeable high-quality dataset of trajectories exists. However, the reality is that immense manual task-specific modeling and engineering are required – something we call *environment shaping* (a generalization of the term reward shaping to include other environment choices to ease optimization) – to make things work, which is the primary bottleneck in our opinion. Although the environment shaping bottleneck is highly relevant to robotics, it exists in any domain where reinforcement learning is applied to real-world problems, including transportation systems (Wei et al., 2018), autonomous driving (Wu et al., 2017), finance (Liu et al., 2021), and power management (Vázquez-Canteli et al., 2019).

We expand the previous usage of the term environment shaping (Co-Reyes et al., 2020) to include all choices such as designing reward function, curriculum, observation/action spaces, initial state distribution, reset functions performed manually to make training possible. These issues have been individually studied for a long time (Selfridge et al., 1985; Ng et al., 1999; Singh et al., 2009; Eßer et al., 2023), but a holistic and critical analysis of the human effort required to make these choices even with the latest algorithmic advances has not been made.

Using examples from recent applications of RL to robotics, **this position paper argues that the primary bottleneck for scaling up reinforcement learning is its need for manual environment shaping.** Specifically, this is a call to action for the RL research community:

- Distinguish *modeling* from *shaping* design choices in RL environments. Many works describe the final shaped environment but not a repeatable procedure that can transfer the same shaping to a new robot or task.
- Prioritize research into *automatic environment shaping* as a pathway to generalize domain-specific successes in RL.
- Prioritize *benchmarks* that measure the total expense of applying reinforcement learning to real-world robotics tasks, by including environments with 'unshaped' versions and corresponding human-shaped baselines. The point being that existing benchmarks have already performed environment shaping (i.e., hide the true problem) and, therefore do not serve as good candidates for further research into better algorithms. [1]

2. Robotic Behavior Generation with RL

To support a precise definition of environment shaping, we first describe a typical workflow for generating robotic behaviors using reinforcement learning (RL) in simulation (Figure 1).

We decompose behavior generation into four subtasks; sample environment generation (Sec 2.1), environment shaping (Sec 2.2), RL training (Sec 2.3), and the outer feedback loop with behavior evaluation and reflection (Sec 2.4). In delineating these substages, we pinpoint typical human efforts involved in the process.

2.1. Modeling Sample Environments

Consider $\hat{p}(e)$ as the oracle distribution of the target environment we want to deploy our robots in. Our goal is to generate behavior that is performant (with respect to objective J) and robust under $\hat{p}(e)$,

$$\max_\pi \mathbb{E}_{\hat{e}\sim\hat{p}(e)} J(\pi; \hat{e})$$

The target environment can either be a specific real-world environment that already exists (e.g. kitchen at a specific location) or a generic concept (e.g. typical household kitchen).

Unfortunately, it's extremely difficult to model this oracle

[1] https://auto-env-shaping.github.io/

Figure 2. Example of environment complexity: an overloaded and disorganized real-world dishwasher.

distribution either way; it requires comprehensive knowledge of all possible environmental variables and conditions, which is often infeasible due to the complexity, variability, and limited observability in the real world. Innate vagueness of generic concepts is often an issue as well. Just imagine modeling a true oracle distribution of a chaotic real-world dishwasher in simulation! (Figure 2)

In contrast, modeling a *single* sample environment, $\hat{e}$, a specific instance drawn from oracle environment distribution,

$$\hat{e} \sim \hat{p}(e),$$

and importing that to a simulation is much more feasible. This is why robotics practitioners typically start the behavior generation process by first designing a single sample environment: (a) modeling and importing robots and necessary assets in simulation, and (b) manually placing them in their default poses. We often generate a *set* of those sample environments to kick things off. This is what practitioners do for the first blue box in Figure 1.

Such a set of sample environments actually serves a purpose: it is a useful representative testbed environment that can be used to estimate the behavior performance under the true oracle distribution $\hat{p}(e)$. Combined with any form of task specification r (Agrawal, 2022), we define a simulated testbed $\mathcal{E}^{\text{test}}$ where the trained behaviors can be evaluated.

Definition 2.1 (Test Environment)**.** Let $\{\hat{e}_1, \cdots, \hat{e}_n\}$ be a set of n *sample environments* each independently drawn from oracle environment distribution $\hat{p}(e)$. The simulated counterparts are denoted as $\hat{e}_{\text{sim},i}$. Given a task specification r, a test environment $\mathcal{E}^{\text{test}}$ is defined as a set of tuples:

$$\mathcal{E}^{\text{test}} = \{\langle \hat{e}_{\text{sim},i}, r \rangle\}_{i=1,\cdots,n},$$

where the generated behavior π will be evaluated in.

Meanwhile, to prevent the behavior from overfitting to a few sample environments within $\mathcal{E}^{\text{test}}$, one can maintain two sets of sample environments; the test environment can be strictly held out from the rest of the behavior generation pipeline, letting it serve the sole purpose of behavior evaluation. A distinct set of sample environments then can be used to effectively guide the remaining design choices. We call that a *reference* environment, $\mathcal{E}^{\text{ref}}$, as illustrated in Figure 1.

Definition 2.2 (Reference Environment)**.** A Reference Environment $\mathcal{E}^{\text{ref}}$ is a distinct set of sample environments that provides useful context to the shaping algorithm. Trained behaviors will not be evaluated here to avoid overfitting.

For instance, when designing a robotic behavior for unloading a dishwasher, sample environments would include multiple instances of dishwashers loaded with varying configurations of dishes and utensils. Sampled configurations then could be split into a set of reference environments for guiding the shaping and a set of held-out test environments for evaluation. This diversity in configurations provides the shaping algorithm with a broad context, incentivizing it to infer the underlying distribution of object placements. The goal is for the reinforcement learning (RL) trainer to sample from this inferred distribution during training, ensuring that the generated behaviors are robust and adaptable to various real-world scenarios without overfitting to a specific set of environments.

2.2. Shaping Reference Environments

A straightforward subsequent step of behavior generation might be to directly use the reference environments $\mathcal{E}^{\text{ref}}$ as an RL environment, with the expectation that algorithms like Proximal Policy Optimization (PPO) (Schulman et al., 2017) will find performant and generalizable behaviors. However, this approach often falls short due to the inherent sparsity of these environments.

Reference environments often present a challenging optimization landscape for RL algorithms due to their *sparse* nature. For example, it might be rare to obtain nonzero rewards or the observation space might be dominated by spurious features that encourage poor local minima. To mitigate these challenges, human engineers typically go through a process of *shaping* the elements of $\mathcal{E}^{\text{ref}}$. This modification, aimed at enhancing the *learnability* of the environment, includes introducing denser learning signals and additional modifications encouraging effective exploration. The resulting *shaped environment* (Co-Reyes et al., 2020) then used as a training ground for the subsequent optimal control solver.

Definition 2.3 (Shaped Environment)**.** A *Shaped Environment* $\mathcal{E}^{\text{shaped}}$ is a modification of a reference environment, i.e., $\mathcal{E}^{\text{shaped}} = f(\mathcal{E}^{\text{ref}})$. The transformation f incorporates design choices specifically optimized for learning performance, smoothing the optimization landscape enabling the solver to find better solutions with maximal performance in

$\mathcal{E}^{\text{test}}$. The transformation function

$$f : \mathcal{E}^{\text{ref}} \to \mathcal{E}^{\text{shaped}} \tag{1}$$

is defined as *shaping function*.

The *shaping* process for a robotics environment usually involves manually explored design choices, including shaping the reward (Ng et al., 1999; Singh et al., 2009), modifying the action space (Peng & Van De Panne, 2017; Aljalbout et al., 2023), designing curricula on the environment dynamics, initial state distributions, and goal distributions (Selfridge et al., 1985; Matiisen et al., 2019; Portelas et al., 2020; Lee et al., 2020), crafting the state space (Yu et al., 2023), and shaping the right failure conditions for early termination (Co-Reyes et al., 2020). We describe details of common shaping operations in Sec 3.1.

2.3. RL Training

Once the shaped environment is obtained, the next step of behavior generation is to use RL algorithms, i.e., PPO (Schulman et al., 2017), to find a behavior π that best performs on the shaped environment $\mathcal{E}^{\text{shaped}}$. Formally, the algorithm aims to find an optimal behavior π for the following optimization problem:

$$\begin{aligned} &\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t(s_t,a_t))\right] \\ &\text{s.t.} \quad s_{t+1} \sim p(s_t, a_t; \mathcal{E}^{\text{shaped}}). \end{aligned} \tag{2}$$

While the RL training process also requires a range of design decisions, such as algorithmic choices and hyperparameter adjustments, these areas are relatively well-researched and documented (Parker-Holder et al., 2022; Kiran & Ozyildirim, 2022).

However, we note that in a pratical context of robotic behavior generation, tuning the RL setting (e.g. neural architecture search for policy or hyperparameter tuning) is often underprioritized compared to the effort put into environment shaping. In IsaacGymEnvs (Makoviychuk et al., 2021), for instance, simple Multilayer Perceptron (MLP) networks are employed, and off-the-shelf RL algorithm implementations are utilized with their default configurations. This shows that algorithms like PPO and their default settings are capable enough when paired with *ideally shaped* environment.

2.4. Optimizing Environment Shaping via Iterative Behavior Evaluation and Reflection

Once an optimal behavior $\pi^\star$ is obtained via RL training, the behavior is *evaluated* on the test environment $\mathcal{E}^{\text{test}}$ and *reflected* by human engineers. Denoting the reflection process as $\mathcal{H}$ of analyzing the generated behavior $\pi^\star$ in test environment $\mathcal{E}^{\text{test}}$ and coming up with a better environment shaping f,

$$\mathcal{H} : f_k \times J(\pi_k^\star; \mathcal{E}^{\text{test}}) \to f_{k+1}, \tag{3}$$

robotic behavior generation process can be formally defined as an **iterative optimization process** over the **environment shaping function** f,

$$\begin{aligned} &f_{k+1} = \mathcal{H}\left(f_k, J(\pi_k^\star; \mathcal{E}^{\text{test}})\right) \\ &\text{where} \;\; \pi_k^\star = \operatorname*{argmax}_{\pi} J(\pi; \mathcal{E}_k^{\text{shaped}}), \\ &\qquad\qquad \mathcal{E}_k^{\text{shaped}} = f_k(\mathcal{E}^{\text{ref}}), \;\; f_0 = \mathbf{I}_{\text{identity}} \end{aligned} \tag{4}$$

which aims to find an optimal shaping function $f \in \mathcal{F}$ for the following bi-level optimization problem:

$$\begin{aligned} &\max_{f\in\mathcal{F}} J(\pi^\star; \mathcal{E}^{\text{test}}) \\ &\text{s.t.} \;\; \pi^\star \in \arg\max_{\pi} J(\pi; \mathcal{E}^{\text{shaped}}), \;\; \mathcal{E}^{\text{shaped}} = f(\mathcal{E}^{\text{ref}}). \end{aligned} \tag{5}$$

After shaping is applied, the new environment may not reflect the original task; therefore, note that the outer level of the bilevel optimization maximizes $J(\pi^*; \mathcal{E}^{\text{test}})$ which is the return evaluated in the original test environments without any shaping. If the inner level optimizes a shaped environment well, but with poor correspondence to the original task, it will be dispreferred by the outer loop.

3. The Current State of Environment Shaping

Having established the role of environment shaping operations in successful behavior generation, we now probe deeper into the unique challenges that the problem of environment shaping and its optimization procedure presents. Specifically, we make the following arguments with supporting experiments and analysis:

- Popular RL benchmark environments are *artificially* made easy for RL with task-specific environment shaping. We should benchmark our algorithms in unshaped environments if we want them to solve new problems without task-specific environment shaping step. (Section 3.1)
- Shaping multiple attributes of the environment at once (reward, observation space, action space, etc.) is a tricky, non-convex optimization problem (Section 3.2).
- Reward shaping is not the only problem. Existing automation efforts focus too narrowly on rewards (Section 3.3).

3.1. RL Benchmarks for Robotics are Artificially Easy

Benchmark environments for robot reinforcement learning include task-specific environment shaping to make it feasible to help baseline RL algorithms perform reasonably

AllegroHand	Reward	Change
all shaped	38777	–
sparse reward	0	↓ 38777
unshaped action space	21530	↓ 17247
unshaped observation space	2114	↓ 36663
no early termination	0	↓ 38777
single initial state	0	↓ 38777
single goal state	141155	↑ 102378

Humanoid	Reward	Change
all shaped	7554	–
sparse reward	5237	↓ 2317
unshaped action space	67	↓ 7487
unshaped observation space	0	↓ 7554
no early termination	705	↓ 6849
single initial state	5735	↓ 1819

Anymal	Reward	Change
all shaped	−45	–
sparse reward	−2789	↓ 2744
unshaped action space	−2499	↓ 2454
unshaped observation space	−2656	↓ 2611
no early termination	−43	↑ 2
single initial state	−17	↑ 28
single goal state	−2516	↓ 2470

Table 1. **Impact of environment shaping on policy optimization.** Removing task-specific design choices in the reward, action space, state space, early termination, or initialization incurs performance reductions. Top row: environment with original *shaped* design choices. Each subsequent row shows performance after training with a corresponding unshaped design choice. The performance of all policies is evaluated in a fully unshaped environment.

```python
def shaped_action_space(self):
  # scale the targets by the joint limits
  self.cur_targets = scale(self.actions,
         self.shadow_hand_dof_lower_limits,
         self.shadow_hand_dof_upper_limits)

  # compute the moving average of targets
  self.cur_targets = self.alpha * self.cur_targets +
                (1 - self.alpha) * self.prev_targets
  self.cur_targets = tensor_clamp(self.cur_targets,
              self.shadow_hand_dof_lower_limits,
              self.shadow_hand_dof_upper_limits)
  self.prev_targets = self.cur_targets[:]

  # compute the torques according to PD control law
  torque = 3 * (self.cur_targets - self.
    shadow_hand_dof_pos) - 0.1 * self.
    shadow_hand_dof_vel

  # apply torques
  self.gym.set_dof_actuation_force_tensor(self.sim,
    gymtorch.unwrap_tensor(torque))
```

```python
def unshaped_action_space(self):
  # directly apply the prediction action as torques
  self.gym.set_dof_actuation_force_tensor(self.sim,
    gymtorch.unwrap_tensor(self.actions))
```

Figure 3. Action space shaping: (Top) Original shaped action space with task-specific features. (Bottom) Unshaped action space consisting of joint torque commands. Some shaped code has been slightly modified from the source to increase brevity and clarity while preserving the original logic.

well. However, these modified environments might not fully assess how RL algorithms progress towards solving various control problems independently, treated as a black-box, without needing task-specific adjustments.

In this section, we outline the common task-specific design choices associated with different aspects of environment shaping, while formally defining what an *unshaped* counterpart — one with minimal or no human engineering required — would look like. We consider a case study of the IsaacGymEnvs task suite (Makoviychuk et al., 2021).

Action Space. How would an *unshaped* action space look, and how *shaped* is the action space in the case environments?

For a typical robot in rigid multibody simulation, the unshaped action space would be the motor torques: passing in the policy output a directly to the motor as a torque τ — no scaling or transforming the outputs, no gains to be tuned.

Designing a shaped action space for a robot is thus equivalent to the problem of choosing a low-level controller (and its corresponding parameters) that converts a policy output $\mathbf{a}$ with different physical meanings into an executable motor torque τ that can drive the actuators. The low-level controller can implement a prior like whether to resist or comply to external forces. Examples include proportional-derivative control, differential inverse kinematics controller, operational space control (Khatib, 1985), or impedance control (Hogan, 1984). These are projections of the policy outputs onto a strict subset of the space of possible torque commands. The choice of action space can significantly influence the performance of learning algorithms in the environment (Peng & Van De Panne, 2017).

In IsaacGymEnvs, there are multiple distinct cases of shaped action spaces. `AllegroHand` uses joint position targets with moving average smoothing as its action space (See Figure 3 for example code). For `Anymal`, the relative joint position target is used as an action space with proportional-derivative controller. For `Humanoid`, scaled joint torques are used. Table 1 empirically shows that removing these *shaping* operations on action space largely impacts the training performance.

Observation Space. What is an *unshaped* observation space, and how did IsaacGymEnvs *shape* the observation?

Crafting a *shaped* observation space involves strategic selection and transformation of variables from an unshaped observation — the entire state in simulation that's been ex-

```python
def shaped_observation_space(self):
  root_states = rigid_bodies[:, self.root_handle]
  base_quat = root_states[:, 3:7]
  # base linear velocity (from local frame)
  base_lin_vel = quat_rotate_inverse(base_quat,
    root_states[:, 7:10]) * lin_vel_scale
  # base angular velocity (from local frame)
  base_ang_vel = quat_rotate_inverse(base_quat,
    root_states[:, 10:13]) * ang_vel_scale
  # transformed base orientation
  projected_grav = quat_rotate(base_quat, gravity_vec)

  # scaling/normalizing values
  commands_scaled = scale(commands,
                  [lin_vel_scale, ang_vel_scale])
  dof_pos_scaled = dof_pos_scale *
                  (dof_pos - default_dof_pos)
  dof_vel_scaled = dof_vel_scale * dof_vel

  # concatenate the relevant features
  obs = torch.cat([base_lin_vel, base_ang_vel,
                   projected_grav, commands_scaled,
                   dof_pos_scaled, dof_vel_scaled,
                   actions], dim=-1)
```

```python
def unshaped_observation_space(self):
   # include entire unprocessed simulator state
   obs = torch.cat([dof_pos,
               dof_vel, torques,
               rigid_bodies.reshape(num_envs, -1),
               rigid_body_force.reshape(num_envs, -1),
               commands, actions], dim = -1)
```

Figure 4. State space shaping: (Top) Original shaped state space with task-specific features. (Bottom) Unshaped state space contains the entire raw simulator state.

posed to the user. The process, integral to optimizing policy learning performance, includes (a) useful transformation of the unshaped variables (i.e., features) and (b) discarding unnecessary variables. Figure 4 shows the details of the shaped state space in the IsaacGymEnvs `Anymal` environment.

Different choices of observation space shaping can substantially impact robot performance. In the setting of quadrupedal locomotion, Yu et al. (2023) found that different aspects of the task (robustness, performance) are sensitive to different features included in the observation space. Table 1 shows our findings agreeing that observation shaping has a significant impact on training performance. For `Humanoid`, for instance, using an unshaped observation space makes the task completely unlearnable.

Reward. The *unshaped* reward represents the true objective function we actually wish to optimize, often defined as extrinsic reward (Singh et al., 2010; 2009) or task-fitness function (Niekum et al., 2010).

Shaping the *extrinsic* reward into a more informative *intrinsic* reward to make it more conducive to RL algorithms is a well-studied topic in the community (Ng et al., 1999; Singh et al., 2010; Gupta et al., 2022; Eschmann, 2021). The shaping procedure is usally designed to provide *denser* learning signals and to prevent the policy from overly exploiting the innate vagueness of extrinsic reward, i.e., reward hacking. It typically involves a few commonly used strategies. One popular strategy is to reward *progress* towards the goal by adding distance metrics measuring how close the agent is to reaching it. If some attributes of successful behavior are known beforehand, those terms can be added to the shaped reward (Chen et al., 2020; Margolis & Agrawal, 2023). If a trajectory of a successful behavior is known, one can also compute a *similarity* between the generated behavior and the trajectory and try to maximize the similarity (Peng et al., 2021). Offering bonuses for exploring new states (Pathak et al., 2017) is also known to facilitate RL training. Table 1 shows that *shaping* reward has significant impact on the training performance.

It's worth noting, however, that defining the extrinsic reward itself poses its own set of challenges, which is well addressed in (Agrawal, 2022). The task fundamentally requires translating human intentions and goals into numerical values that an algorithm can optimize, often posing significant challenges. In the scope of this position paper, we presume the task has been defined and consider the challenge of learning a policy to perform it.

Initial/Goal State. Let's consider a concrete example: how would you *shape* an initial state of a dishwasher to make the behavior robust under the chaotic randomness typically exhibited in real-world (Figure 2)? Where would you even start shaping things from? How would an *unshaped* counterpart look like for initial/goal state shaping?

A reasonable *unshaped* intial/goal state to start with are the nominal states defined in *reference* environment – manually designed sample environment with every actors (robots and assets) staying in its nominal pose. We often assume such nominal pose to be a mean of its underlying distribution, commonly assumed as Gaussian or Uniform distribution. This is indeed the most simple yet common technique of designing initial state distribution for many robotics tasks – we just randomly perturb robot joints and assets around its nominal (reference) pose!

When the task gets more complex, however, this simple approach starts to break quickly. Imagine randomly perturbing a single nominal state of a dishwasher, or randomly perturbing the nominal pose of a quadruped standing on a rough terrain; dishes and ladles, feets and terrain will be in penetration most of the time. Doing rejection sampling can be a stopgap solution, but it might end up rejecting most of the samples, making the approach nearly unusable.

In practice, robotics engineers thus take a clever, but heavily heurstic, task-dependent approach to shape initial states. To randomly initialize a quadruped on a rough terrain, for instance, people make the robot walk off a small region of flat ground (Rudin et al., 2022), letting the simulation engine figure out the phsyics constraints. To generate random

initial states for cluttered bin-picking tasks, we often drop objects from height in random order. To obtain initial states to train a fall-recovery policy for humanoid, we drop the humanoid from height (Peng et al., 2021). For in-hand manipulation tasks, to obtain a diverse set of downward-facing initial grasps to start with, we first train a grasping policy and execute it to generate diverse initial starting configurations that does not drop the object immediately (Chen et al., 2022b). The task-specific nature of such strategies poses challenges in automating this shaping operation.

Moreover, designing *how* we sample from the shaped initial/goal distribution can naturally set up a learning curriculum that progresses from simpler to more challenging tasks. Take the example of training a quadruped to run at high-speed. Training only with high-speed commands could hinder the learning process. Instead, shaping a goal distribution to be a wider distribution then the actual target speed, and designing the sampling strategy to begin with slower ones and gradually increasing, can facilitate more effective learning of fast running behaviors (Margolis et al., 2022).

Terminal Condition. A strict definition of *unshaped* terminal condition might be to never terminate, resembling how the real world never terminates and never gets a chance to reset from the beginning. However, since this poses quite challenging problem for most learning algorithms, one more common choice of *unshaped* terminal condition that is fairly environment agnostic is to set a predefined episode time limit.

Shaping the terminal condition thus corresponds to deploying strategies of *early termination*, which is known to significantly improve training performance (Co-Reyes et al., 2020). In IsaacGymEnvs, `Anymal` terminates when non-foot bodies contact the ground, `Humanoid` terminates when the torso falls below a certain height, and `AllegroHand` terminates when the object falls off from the hand. Table 1 shows that early termination significantly improves the training performance in all three tasks.

3.2. Shaping the Entire Environment is Harder than Shaping One Component

In the previous section, we saw that policy learning can be highly sensitive to each individual aspect of environment shaping (Table 1) and that the shaping choices in IsaacGymEnvs vary qualitatively depending on the task. This motivates that, to promote the development of truly automatic behavior generation, we should consider benchmarking against a suite of unshaped environments. If an algorithm can learn policies in unshaped environments, it should be applicable to newly defined tasks and environments without requiring additional manual shaping on those environments. The unshaped environments are an appropriate benchmark for methods that couple automatic environment shaping with RL, or to directly attempt to solve by improving RL algorithms.

What kinds of methods might we try using to optimize an entirely unshaped environment? If we aim to accomplish this by introducing shaping, we may have to search over not just one aspect but all aspects of shaping. To understand the optimization landscape, suppose we have access to an oracle that proposes the human-designed environment shaping as a candidate, and we use this to perform a hill-climbing search in one aspect of the environment at a time: first, optimize the observation space, then the action space, then the reward function, and repeat until convergence. Figure 5 illustrates the consequence of this strategy in three environments from IsaacGymEnvs. Each node represents a shaped version of the environment. The node's color indicates the trained policy's performance on the shared test environment (unshaped). Edges indicate environments that differ by one type of change (state space, reward, etc.), and the bold arrows indicate the path taken by decoupled hill-climbing on one change at a time. We found that each environment has multiple local maxima where the hill climbing would get stuck in a suboptimal configuration. These local maxima differ in at least two (as many as four!) types of shaping. This suggests that the environment shaping design space is non-convex in the different types of shaping. Therefore, we should aim to develop techniques that consider shaping all parts of the environment jointly.

3.3. Existing Automation Focuses Narrowly on Rewards

There have been prior attempts to partially automate the process of environment shaping. However, most of the efforts have focused on the subtask of reward shaping. The concept of reward shaping was formalized by (Ng et al., 1999) and the idea of automated reward shaping through evolutionary search was advanced by Singh et al. (2009; 2010). Recent works have formulated the problem of reward shaping as bilevel optimization with the environment design at the top level and policy learning at the bottom level and used LLMs (Goyal et al., 2019; Ma et al., 2023; Xie et al., 2023) or gradient-based methods (Hu et al., 2020) to generate candidates.

LLM-based methods can benefit from prior knowledge about coding and robotics derived from internet training data to act as an efficient sampler for generating candidate shaped rewards expressed as code. Since other aspects of the environment shaping can also be expressed as code, it is straightforward to test how these methods extend. We evaluated the LLM-based reward shaping algorithm Eureka (Ma et al., 2023) at the task of designing the observation and action space for the `Anymal` IsaacGymEnvs environment. As in the original application to reward shaping, we generated five generations of 16 parallel candidate shaping functions

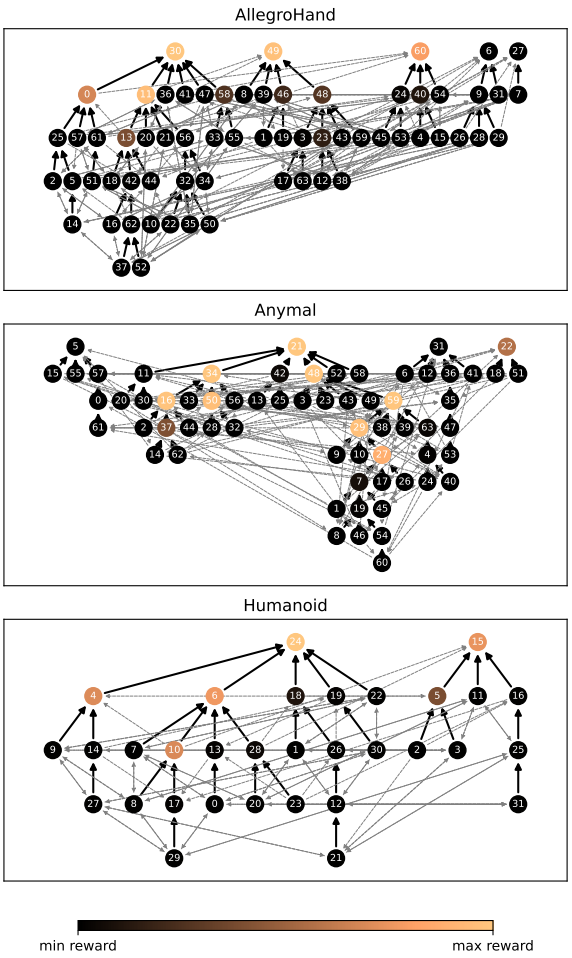

Figure 5. **Local optima in environment shaping problems.** Each node represents a shaped training environment. Edges connect environments that are separated by modifying one type of shaping (action space, state space, reward function, initial state, goal, or terminal condition). Bold arrows represent optimal choices for hill climbing. Each environment is shown to have multiple local optima corresponding to the top row of nodes.

using GPT-4, including the best performing candidate from each generation in the prompt for the next generation. Eureka succeeded in designing effective observation and action spaces for this environment (Table 2) while all other aspects of the environment were pre-shaped.

Motivated by Section 3.2, we also tested whether Eureka can jointly optimize multiple aspects of the environment. We found that performance drastically drops when Eureka is tasked to jointly optimize multiple environment aspects, i.e., shaping both the reward and observation, even though it could optimize each individually. This suggests that shaping multiple environment aspects jointly is yet an open problem.

4. Paths Forward to Automated Environment Shaping

Today, no algorithm can solve diverse unshaped tasks, although we know them to be solvable with environment shaping by human designers. How can we fix this and generalize successes in RL for robotics? There are a few possibilities:

Scale up computation. Existing bi-level search techniques like Ma et al. (2023) can be extended to design environment shaping and run with more compute resources to search over a greater number of candidate shaping designs. Massively parallel simulation has led to realistic robotics environments that can train quickly (Makoviychuk et al., 2021; Rudin et al., 2022); However, some state-of-the-art sim-to-real methods take weeks to train a single policy due to high data requirements or expensive subroutines (Chen et al., 2023; Jenelten et al., 2024). This would make performing much more outer-loop search impractical.

Improve priors. If we can't search over more candidates, a better way is to generate higher-quality candidates more quickly. One possibility is that improved foundation models will zero-shot generate better candidate shaped environments (Xian et al., 2023; Yang et al., 2023). However, it's hard to predict how much this will help. Another good idea is to *mechanistically understand the strategies of human designers*. By what mechanism does the choice of observation space or curriculum improve performance? Co-Reyes et al. (2020) proposed a holistic view of environment shaping ('Ecological Reinforcement Learning') and studied the impact of stochasticity, goal distribution, and early termination design on learnability. Yu et al. (2023) examined the impact of observation space on learning quadruped locomotion and offered an explanation of how different observations can work better for different subtasks. Eßer et al. (2023) surveyed a number of works across applied reinforcement learning, comparing practitioners' design choices. Peng & Van De Panne (2017) analyzed a set of common action spaces for robot control and Aljalbout et al. (2023) developed metrics to explain the performance gap. Kim et al. (2023) found that constraints require less tuning than rewards to transfer across diverse robots. If more understanding is documented in these areas, improved biases could be implemented as a prior, e.g., in an LLM's context.

Shape online. Instead of iteratively improving on the environment shaping f across training runs, can we improve it dynamically within a RL training loop? If, for example, multi-objective reinforcement learning or bilevel optimization can trade off the coefficients of multiple reward terms (Singh et al., 2010; Zheng et al., 2018; Chen et al., 2022a), then perhaps the LLM search can be performed only over the form of the different reward terms, and their relative weighting can be optimized at runtime. In other aspects of the environment, an analogous approach would

shaping component	Eureka (Ma et al., 2023)	Human Design (Makoviychuk et al., 2021)	Automation Performance
*reward	0.986	0.973	$\uparrow$ 0.013
†observation	0.967	0.973	$\downarrow$ 0.006
†action	0.982	0.973	$\uparrow$ 0.009
$^{\circ}$reward $\times$ observation	0.196	0.973	$\downarrow$ 0.777
$^{\circ}$reward $\times$ action	0.536	0.973	$\downarrow$ 0.437
$^{\circ}$reward $\times$ observation $\times$ action	N/A	0.973	N/A

Table 2. Evaluating (Ma et al., 2023) for *reward shaping, †shaping different components, and $^{\circ}$coupled shaping. `Anymal` task (Makoviychuk et al., 2021) is used as an environment. Automation performance is measured by the relative gain obtained by automating the shaping process, compared to design choices made by humans (Makoviychuk et al., 2021). Behavior is evaluated with the following task specification: $r = \exp(\text{negative\_distance\_to\_command})$. 5 iterations of outer loop for Eureka. Used latest GPT-4 model. Note that GPT-4 failed to generate working code for results labeled N/A.

be to generate a parameterized shaping operator and automatically tune its parameters. For example, the scale of the observation or the termination height could be assigned as parameters for online optimization.

A Robotics Benchmark for Environment Shaping. Instead of evaluating RL algorithms on environments preshaped to work with PPO, the community should evaluate them on unshaped environments. These environments will be *too hard* to solve with existing RL algorithms. Thus, it will be necessary to (a) study how to modify aspects of the environment to make it efficiently solvable with RL and / or (b) develop better RL algorithms that can handle such challenging sparse environments.

We modified the environments in IsaacGymEnvs to use unshaped design choices to provide a good proxy for the task-agnostic output from procedural environment generators. To facilitate environment shaping research, our code exposes an API for modifying the environment code, which allows the optimizer to transform the reward, observation space, action space, etc. by editing Python functions at runtime. The API is designed so that any language models can be easily integrated to perform such transformations. Our implementation also facilitates faster evaluation of multiple environment shaping choices by training multiple policies in a single process, leveraging parallel simulation.

5. Conclusion

Reinforcement learning has long promised to solve decision-making problems in a task-agnostic manner. It has found great success in solving challenging but narrowly-scoped tasks in robotics. In this position paper, we argued that the key bottleneck for scalability of RL is a limited mechanistic understanding of task-specific engineering (*environment shaping*) that transforms environments to be solved more easily and is universal across domains and benchmarks. We proposed a formal definition of environment shaping as an optimization problem and identified instances of shaping in robotic tasks. Finally, we identified key steps forward such as developing computationally efficient search over environment shaping; improving our tools for implementing such shaping and understanding its impact on learning dynamics; and defining benchmarks for this problem. We hope this will motivate an increased focus in RL research on communicating and evaluating environment-shaping measures that impact performance rather than solely emphasizing the impact of the learning algorithm or policy architecture.

Acknowledgements

We thank the members of the Improbable AI lab for the helpful discussions and feedback on the paper. We are grateful to MIT Supercloud and the Lincoln Laboratory Supercomputing Center for providing HPC resources. This research was partly supported by Hyundai Motor Company, DARPA Machine Common Sense Program, the MIT-IBM Watson AI Lab, and the National Science Foundation under Cooperative Agreement PHY-2019786 (The NSF AI Institute for Artificial Intelligence and Fundamental Interactions, http://iaifi.org/). We acknowledge support from ONR MURI under grant number N00014-22-1-2740. Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-21-1-0328. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agrawal, P. The task specification problem. In *Conference on Robot Learning*, pp. 1745–1751. PMLR, 2022.

Aljalbout, E., Frank, F., Karl, M., and van der Smagt, P. On the role of the action space in robot manipulation learning and sim-to-real transfer. *arXiv preprint arXiv:2312.03673*, 2023.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Chen, D., Zhou, B., Koltun, V., and Krähenbühl, P. Learning by cheating. In *Conference on Robot Learning*, pp. 66–75. PMLR, 2020.

Chen, E., Hong, Z.-W., Pajarinen, J., and Agrawal, P. Redeeming intrinsic rewards via constrained optimization. *Advances in Neural Information Processing Systems*, 35: 4996–5008, 2022a.

Chen, T., Xu, J., and Agrawal, P. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pp. 297–307. PMLR, 2022b.

Chen, T., Tippur, M., Wu, S., Kumar, V., Adelson, E., and Agrawal, P. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8 (84):eadc9244, 2023.

Cheng, X., Shi, K., Agarwal, A., and Pathak, D. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023.

Co-Reyes, J. D., Sanjeev, S., Berseth, G., Gupta, A., and Levine, S. Ecological reinforcement learning. *arXiv preprint arXiv:2006.12478*, 2020.

Eschmann, J. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications*, pp. 25–33, 2021.

Eßer, J., Bach, N., Jestel, C., Urbann, O., and Kerner, S. Guided reinforcement learning: A review and evaluation for efficient and effective real-world robotics [survey]. *IEEE Robotics & Automation Magazine*, 30(2):67–85, 2023. doi: 10.1109/MRA.2022.3207664.

Fu, Z., Zhao, T. Z., and Finn, C. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

Goyal, P., Niekum, S., and Mooney, R. J. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.

Gupta, A., Pacchiano, A., Zhai, Y., Kakade, S. M., and Levine, S. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity, 2022.

Handa, A., Allshire, A., Makoviychuk, V., Petrenko, A., Singh, R., Liu, J., Makoviichuk, D., Van Wyk, K., Zhurkevich, A., Sundaralingam, B., et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5977–5984. IEEE, 2023.

Hoeller, D., Rudin, N., Sako, D., and Hutter, M. Anymal parkour: Learning agile navigation for quadrupedal robots. *arXiv preprint arXiv:2306.14874*, 2023.

Hogan, N. Impedance control: An approach to manipulation. In *1984 American control conference*, pp. 304–313. IEEE, 1984.

Hu, Y., Wang, W., Jia, H., Wang, Y., Chen, Y., Hao, J., Wu, F., and Fan, C. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941, 2020.

Jenelten, F., He, J., Farshidian, F., and Hutter, M. Dtc: Deep tracking control. *Science Robotics*, 9(86):eadh5401, 2024.

Ji, Y., Margolis, G. B., and Agrawal, P. Dribblebot: Dynamic legged manipulation in the wild. *International Conference on Robotics and Automation*, 2023.

Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., and Scaramuzza, D. Champion-level drone racing using deep reinforcement learning. *Nature*, 620 (7976):982–987, 2023.

Khatib, O. The operational space formulation in the analysis, design, and control of robot manipulators. In *Preprints 3rd International Symposium of Robotics Re-search, Gouvieux (Chantilly), France, October*, pp. 7–11, 1985.

Kim, Y., Oh, H., Lee, J., Choi, J., Ji, G., Jung, M., Youm, D., and Hwangbo, J. Not only rewards but also constraints: Applications on legged robot locomotion. *arXiv preprint arXiv:2308.12517*, 2023.

Kiran, M. and Ozyildirim, M. Hyperparameter tuning for deep reinforcement learning applications. *arXiv preprint arXiv:2201.11182*, 2022.

Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

Lei, K., He, Z., Lu, C., Hu, K., Gao, Y., and Xu, H. Uni-o4: Unifying online and offline deep reinforcement learning

with multi-step on-policy optimization. *arXiv preprint arXiv:2311.03351*, 2023.

Liu, X.-Y., Rui, J., Gao, J., Yang, L., Yang, H., Wang, Z., Wang, C. D., and Jian, G. FinRL-Meta: Data-driven deep reinforcementlearning in quantitative finance. *Data-Centric AI Workshop, NeurIPS*, 2021.

Luo, J., Hu, Z., Xu, C., Tan, Y. L., Berg, J., Sharma, A., Schaal, S., Finn, C., Gupta, A., and Levine, S. Serl: A software suite for sample-efficient robotic reinforcement learning. *arXiv preprint arXiv:2401.16013*, 2024.

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. Eureka: Human-level reward design via coding large language models. 2023.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

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

Margolis, G. B., Yang, G., Paigwar, K., Chen, T., and Agrawal, P. Rapid locomotion via reinforcement learning. *arXiv preprint arXiv:2205.02824*, 2022.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

Miki, T., Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7 (62):eabk2822, 2022.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.

Niekum, S., Barto, A. G., and Spector, L. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.

Parker-Holder, J., Rajan, R., Song, X., Biedenkapp, A., Miao, Y., Eimer, T., Zhang, B., Nguyen, V., Calandra, R., Faust, A., et al. Automated reinforcement learning (autorl): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, 2022.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Peng, X. B. and Van De Panne, M. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–13, 2017.

Peng, X. B., Ma, Z., Abbeel, P., Levine, S., and Kanazawa, A. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

Portelas, R., Colas, C., Weng, L., Hofmann, K., and Oudeyer, P.-Y. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

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

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

Selfridge, O. G., Sutton, R. S., and Barto, A. G. Training and tracking in robotics. In *Ijcai*, pp. 670–672, 1985.

Singh, S., Lewis, R. L., and Barto, A. G. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pp. 2601–2606. Cognitive Science Society, 2009.

Singh, S., Lewis, R. L., Barto, A. G., and Sorg, J. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

Song, Y., Romero, A., Müller, M., Koltun, V., and Scaramuzza, D. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462, 2023.

Vázquez-Canteli, J. R., Kämpf, J., Henze, G., and Nagy, Z. Citylearn v1. 0: An openai gym environment for demand response with deep reinforcement learning. In *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*, pp. 356–357, 2019.

Wei, H., Zheng, G., Yao, H., and Li, Z. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2496–2505, 2018.

Wu, C., Kreidieh, A., Vinitsky, E., and Bayen, A. M. Emergent behaviors in mixed-autonomy traffic. In *Conference on Robot Learning*, pp. 398–407. PMLR, 2017.

Wu, P., Escontrela, A., Hafner, D., Abbeel, P., and Goldberg, K. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning*, pp. 2226–2240. PMLR, 2023.

Xian, Z., Gervet, T., Xu, Z., Qiao, Y.-L., and Wang, T.-H. Towards a foundation model for generalist robots: Diverse skill learning at scale via automated task and scene generation. *arXiv preprint arXiv:2305.10455*, 2023.

Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., and Yu, T. Text2reward: Automated dense reward function generation for reinforcement learning. *arXiv preprint arXiv:2309.11489*, 2023.

Yang, Y., Sun, F.-Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., et al. Holodeck: Language guided generation of 3d embodied ai environments. *arXiv preprint arXiv:2312.09067*, 2023.

Yin, Z.-H., Huang, B., Qin, Y., Chen, Q., and Wang, X. Rotating without seeing: Towards in-hand dexterity through touch. *arXiv preprint arXiv:2303.10880*, 2023.

Yu, W., Yang, C., McGreavy, C., Triantafyllidis, E., Bellegarda, G., Shafiee, M., Ijspeert, A. J., and Li, Z. Identifying important sensory feedback for learning locomotion skills. *Nature Machine Intelligence*, 5(8):919–932, 2023.

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

Zheng, Z., Oh, J., and Singh, S. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.

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