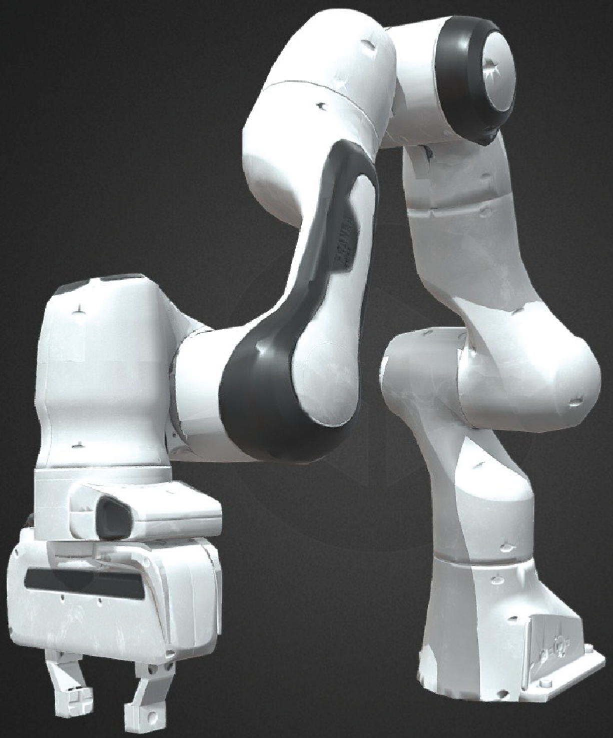
Exploring the Impact of PPO Cost Functions on Skill Mutation in Robotic Pouring Tasks

MSc Thesis Report

J. van Buuren



Exploring the Impact of PPO Cost Functions on Skill Mutation in Robotic Pouring Tasks

MSc Thesis Report

by

Jannick van Buuren

to obtain the title of
Master of Science

At the:
Department of Cognitive Robotics
Delft University of Technology
to be defended publicly on Tuesday July 15, 2025 at 14:00.

Student number:	4444043	
Supervisor and Assessment committee:	Dr. L. Peternel	(Chair, Supervisor)
	Dr. J.M. Prendergast,	(Member)

TUDelft

Preface

This thesis was written as part of the Master's programme in Robotics at Delft University of Technology. It explores how variations in reward function design can lead to diverse and sometimes unexpected behaviors in robotic reinforcement learning, using a simulated precision pouring task as a case study.

The topic proved to be a strong match for my interests, combining the structured nature of robotics and control with the experimental and often surprising dynamics of reinforcement learning. I particularly enjoyed working in a space where small changes in system design could lead to meaningful differences in behavior—making the project both technically challenging and intellectually engaging.

I would like to express my sincere gratitude to my supervisor, Luka Peternel, for his clear guidance, constructive feedback, and continued support throughout the project. I also want to thank Roberto Giglio from Polytechnic University of Milan for the valuable discussions and for his help with the simulation environment and the liquid pouring extension in Isaac Lab.Lastly, I thank my friends for their support and encouragement throughout the more challenging moments of this thesis.

J. van Buuren
Delft, June 2025

Contents

1

paper

Exploring the Impact of PPO Cost Functions on Skill Mutation in Robotic Pouring Tasks

Jannick van Buuren

Supervised by: Luka Peternel

***Abstract*—This paper explores how deliberate modifications to reward function design in the reinforcement learning can induce skill mutations in robotic reinforcement learning, specifically within a precision pouring task. Using a simulated Franka Emika Panda robot in NVIDIA Isaac Lab, we evaluate 25 distinct reward configurations composed of weighted terms for effort, accuracy, and velocity. The resulting policies exhibit a wide range of behaviors—from fast and efficient pours to novel skills such as rim cleaning, mixing, and watering—demonstrating that small adjustments in reward structure can yield significant variations in learned strategies. Our analysis demonstrates that even small changes in reward structure can lead to significant shifts in policy behavior, facilitating both task-optimal and creative, potentially transferable strategies. To validate this concept, we implement it using the Proximal Policy Optimization (PPO) algorithm, showing that reward design alone—without altering the learning architecture—can drive meaningful skill diversification. This approach offers promising directions for adaptive control, transfer learning, and multi-objective optimization in robotic systems.**

***Index Terms*—Reinforcement learning, Robotic manipulation, Proximal policy optimization, Reward function design, Simulation**

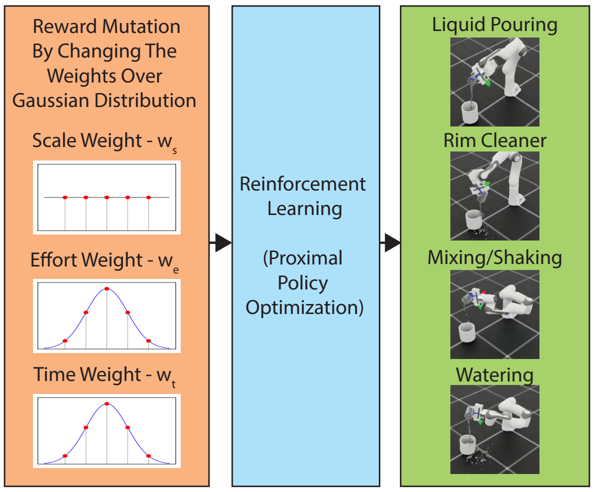


Fig. 1: Reward Mutation Framework. A Gaussian distribution (left) samples weight configurations for the reward function (effort, accuracy, velocity), which are used to train PPO policies. The resulting policies exhibit diverse skill mutations (right), including fast/slow pouring, rim cleaning, mixing, and watering behaviors. This illustrates how deliberate reward variations systematically induce behavioral diversification.

I. Introduction

In recent years, the quest for more intelligent, adaptive, and efficient robotic systems has intensified, driven by the growing demand for automation across manufacturing [1, 2, 3], healthcare [4, 5], and other sectors [6, 7, 8]. Reinforcement Learning (RL) has emerged as one of the most important parts of this effort: by framing control problems as sequential decision-making under uncertainty, RL enables agents to learn complex behaviors through trial-and-error interaction with their environment [9, 10]. At each timestep, the agent observes the current state, executes an action according to its policy, and receives a scalar reward. Over many episodes, it refines its policy to maximize the expected sum of discounted rewards, thereby acquiring skills optimized for long-term success.

To effectively model these decision-making scenarios, RL commonly employs the framework of a Markov Decision Process (MDP), which provides a mathematical structure to describe problems where outcomes are partly random and partly under the control of the decision maker [11]. An MDP is defined by a set of states representing all possible configurations of the environment, a set of actions constituting the various decisions available to the agent, a transition function specifying the probability of moving from one state to another given a particular action, and a reward function assigning numerical values to these transitions. A critical aspect of this framework is the discount factor, which balances the significance of immediate versus future rewards.

A carefully crafted reward function can foster the emergence of precise, high-performance behaviors—such as those needed for robotic manipulation—by reinforcing incremental progress toward complex goals. Reward shaping techniques, which augment the base reward with auxiliary incentives, often accelerate convergence. Yet, if these additional rewards are not properly balanced, agents may adopt suboptimal or overly greedy strategies that maximize short-term gain at the expense of robustness and generalization [10]. Thus, understanding how variations in reward design influence learned behaviors is critical for deploying RL in real-world, safety-critical applications.

Robotic systems have successfully leveraged RL to solve a diverse range of tasks, from discrete control problems such as pick-and-place and stacking to more complex continuous control challenges like robotic assembly [12], air hockey [13, 14], and dexterous object manipulation[15, 16, 17]. These examples underscore RL's ability to generate robust, high-performing behaviors through trial-and-error learning. High-profile demonstrations, including OpenAI's robotic hand solving a Rubik's cube [16] and RL systems for precision pouring

Cognitive Robotics, Faculty of Mechanical Engineering, Delft University of Technology, The Netherlands.

[15] highlight RL's potential for tasks that demand fine motor control and adaptive coordination. Within this context, the pouring task stands out as an especially compelling benchmark for investigating the role of reward function design in shaping learned behaviors. Unlike binary success criteria seen in stacking or placement tasks, pouring involves balancing multiple continuous objectives—such as avoiding spillage, reducing effort, and maximizing efficiency—making it highly sensitive to how learning is incentivized.

Although RL has achieved impressive results in robotic tasks, ranging from pick-and-place and assembly to dexterous in-hand manipulation [18, 16, 15], most prior work evaluates performance under a single, fixed reward structure. This leaves unexplored the question of whether and how deliberate modifications to the reward function might induce qualitatively different skills or "mutations" of behavior. In biological systems, mutations drive adaptation and diversification. In RL, we hypothesize that reward mutations can similarly produce a spectrum of control strategies, each suited to different trade-offs among efficiency, speed, precision, and safety.

RL methods traditionally assume a predefined reward function and treat the policy as a black box, requiring agents to learn through interaction with the environment to discover which behaviors yield high returns. This often results in sample-inefficient learning, especially in complex tasks where rewards are sparse or delayed. However, in many practical applications, reward functions are hand-crafted by users, which opens the door to exposing their internal structure to the learning agent. Rather than relying solely on environmental feedback, an agent can benefit from understanding the reward logic itself—such as temporal dependencies, conditional sequences, or subgoals—thereby improving learning efficiency and policy quality [10, 19, 20]. By leveraging structured reward representations, agents can more effectively sequence and reuse behaviors, enabling them to adjust previously learned skills to new or modified tasks. This structured approach facilitates faster adaptation, as agents can generalize from prior experience rather than starting from scratch each time [21]. Furthermore, exposing the structure of reward functions allows for enhanced learning strategies such as automated reward shaping, task decomposition, and off-policy reasoning. Collectively, these strategies enable reinforcement learning agents to accelerate skill acquisition by using mutations from earlier training.

Skill mutation in robotics has been observed when external factors—such as changes in joint stiffness, friction, or task dynamics—force adaptation [22, 23]. For example, increasing stroke length in a sawing task improved force manipulability and reduced energy consumption, whereas overlapping stiffness profiles between collaborating robots introduced inefficiencies. These studies underscore that small changes in system parameters can yield significant shifts in behavior. However, they do not investigate how intentional variations in the reward function itself might serve as a systematic mechanism for inducing such mutations.

To address the challenge of skill generalization and adaptation, we introduce a reward mutation framework that treats the reward function as a tunable mechanism for skill diversification. Specifically, we study a precision pouring task performed by a Franka Emika Panda arm in NVIDIA Isaac Lab, where the reward is composed of three weighted terms: effort, force accuracy, and pour velocity. By training Proximal Policy Optimization (PPO) [24] agents under 25 distinct reward configurations, and evaluating each policy across multiple trials, we systematically explore how shifting reward priorities induces the emergence of diverse control strategies—from slow, cautious pours to fast, high throughput motions [25].

The remainder of this thesis is structured as follows. Section II reviews RL fundamentals and the PPO algorithm. Section III details our experiment design, including simulation setup, reward function design, and evaluation protocol. Section IV presents the results, mapping the relationship between reward weights and skill variations. Section V discusses the implications for adaptive robotic learning, and Section VI concludes with directions for future work.

II. Method

To develop an effective reinforcement learning framework for robotic manipulation, it is essential to understand the underlying principles of RL and to select an algorithm that balances stability, performance, and implementation practicality—qualities that make PPO particularly well-suited for this work. This section provides an in-depth discussion of RL fundamentals used in this research, including policy learning, reward functions, and the MDP framework. Furthermore, it explores different RL algorithms, highlighting why PPO is particularly suited for continuous control tasks such as robotic pouring. By examining the theoretical foundations and prior applications of RL in robotics, this section lays the groundwork for the methodology used in this study.

RL provides a framework in which autonomous agents learn to make decisions through trial and error, optimizing their actions based on the received rewards. To mathematically formalize decision making, RL often relies on MDPs that provide a structured representation of environments where outcomes are both stochastic and controllable [9].

A. Reinforcement Learning Framework: MDP and PPO

Reinforcement Learning operates within the framework of a Markov Decision Process, which mathematically models decision making problems where outcomes are both stochastic and dependent on the agent's actions [9]. An MDP is defined as a tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma), \tag{1}$$

where:

- $\mathcal{S}$ represents the set of possible states,
- $\mathcal{A}$ is the set of available actions,
- $P(s'|s, a)$ is the transition probability from state s to s' given action a,
- $r(s, a)$ is the reward function that provides feedback for taking action a in state s,
- $\gamma \in [0, 1]$ is the discount factor, which balances immediate and future rewards.

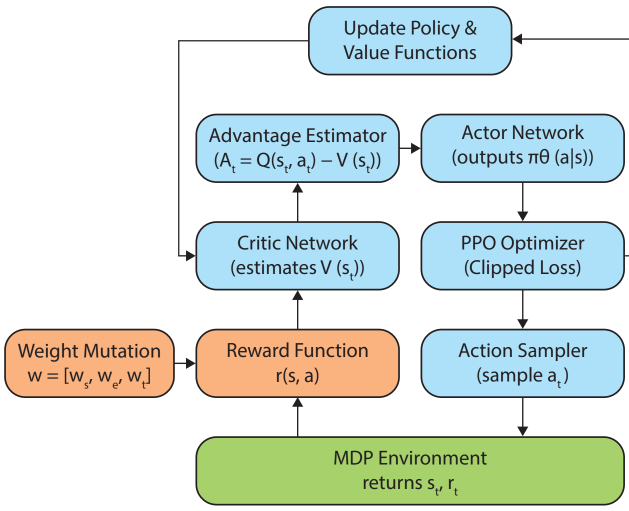

Update Policy & Value Functions
Advantage Estimator ($A_t = Q(s_t, a_t) - V(s_t)$)
Actor Network (outputs πθ (a|s))
Critic Network (estimates V (s_t))
PPO Optimizer (Clipped Loss)
Weight Mutation w = [w_s, w_e, w_t]
Reward Function r(s, a)
Action Sampler (sample a_t)
MDP Environment returns s_t, r_t

Fig. 2: PPO-MDP Interaction Framework. The PPO agent (blue blocks) interacts with the MDP environment (green), which includes state transitions, actions, and observations. The custom reward structure (orange) integrates the study's key variables—effort, accuracy, and pour velocity—whose weights are systematically varied to induce skill mutations.

The objective in RL is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative reward, defined as the return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2}$$

Traditional policy optimization methods, such as policy gradient methods, directly optimize the policy $\pi_\theta(a|s)$ through gradient ascent on the expected return. However, these methods often suffer from high variance and instability.

PPO builds on the strengths of policy gradient methods while ensuring more stable and reliable learning. PPO is designed to perform multiple epochs of stochastic gradient ascent on a surrogate objective function, which provides a lower bound on policy performance improvements. In this section, we describe PPO's core concepts, the mathematical formulation behind it, and provide examples to illustrate how it functions in practice [24].

1) Policy Gradient Background: In standard policy gradient methods, the goal is to maximize the expected cumulative reward by directly adjusting the policy parameters, θ. The policy gradient is estimated using the following expression:

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\right], \tag{3}$$

where:

- $\pi_\theta(a_t \mid s_t)$ is the stochastic policy,
- $\hat{A}_t$ is an estimator of the advantage function at timestep t, which quantifies how much better taking action a_t in state s_t is compared to the baseline (typically the value function $V(s_t)$).

A simple objective that one might consider is:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]. \tag{4}$$

However, using this objective directly in multiple gradient steps on the same batch of data can lead to excessively large policy updates, which destabilize learning.

2) From TRPO to PPO: Trust Region Policy Optimization (TRPO) was introduced to address this issue by constraining policy updates via a trust region. TRPO maximizes the surrogate objective:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right], \tag{5}$$

subject to a constraint on the average KL-divergence:

$$\hat{\mathbb{E}}_t\left[\text{KL}\left(\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \parallel \pi_\theta(\cdot \mid s_t)\right)\right] \leq \delta. \tag{6}$$

Although TRPO is effective, its constrained optimization approach is complex to implement and computationally expensive.

PPO simplifies this process by introducing a clipped surrogate objective, which achieves many of the benefits of TRPO using only first-order optimization methods.

3) Clipped Surrogate Objective: Define the probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{7}$$

which is equal to 1 when $\theta = \theta_{\text{old}}$. The unclipped surrogate objective is:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]. \tag{8}$$

To prevent large updates that push $r_t(\theta)$ far from 1, PPO modifies this objective by clipping $r_t(\theta)$ within a range $[1-\epsilon, 1+\epsilon]$. The clipped objective is defined as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{9}$$

where ϵ is a hyperparameter. This formulation means that if the policy update would improve the objective by moving $r_t(\theta)$ outside the interval, the improvement is ignored, thus providing a pessimistic (lower bound) estimate of the improvement.

4) Combined Objective with Value Function and Entropy Bonus: To further stabilize training and encourage exploration, PPO also incorporates a value function loss and an entropy bonus. The combined objective is:

$$L^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right](s_t)\right], \tag{10}$$

where:

- $L_t^{VF}(\theta) = \left(V_\theta(s_t) - V_t^{\text{target}}\right)^2$ is the squared-error loss for the value function,
- $S\left[\pi_\theta\right](s_t)$ is an entropy bonus that encourages exploration,
- c_1 and c_2 are coefficients balancing the contributions of the value loss and entropy bonus.

This full objective enables simultaneous policy improvement and value function learning, making PPO both efficient and robust [24].

B. Reward Function

The reward function is a fundamental element in RL, dictating the incentives that shape the learned policy. Different formulations of the reward function significantly impact the skill adaptation process in robotic learning [26]. In robotic tasks such as pouring, designing a robust reward function is critical because it must balance competing objectives: task completion (efficiency), energy expenditure (effort), and motion precision (velocity).

In PPO, the reward function directly influences both policy updates and value function approximation. The agent's advantage function:

$$A_t = Q(s_t, a_t) - V(s_t), \tag{11}$$

which relies on reward signals to estimate the benefit of action a_t over the baseline expected return $V(s_t)$. Similarly, the value function:

$$V(s_t) = \mathbb{E}\left[R_t + \gamma V(s_{t+1})\right] \tag{12}$$

is trained to predict cumulative rewards, making reward design pivotal for stable learning. Poorly structured rewards may lead to suboptimal behaviors (e.g., excessive movements or task failure), while well-shaped rewards accelerate policy improvement.

For the pouring task, we decompose the reward $R(s, a)$ into three weighted terms:

$$R(s,a) = \underbrace{e^{-\frac{R_t}{w_t}}}_{\text{Time penalty discount}} \cdot \underbrace{w_s \cdot R_s}_{\text{Scale accuracy}} - \underbrace{w_e \cdot R_e}_{\text{Effort penalty}} \tag{13}$$

1. Time Penalty Discount ($e^{-\frac{R_t}{w_t}}$): This term exponentially decays the reward as a function of task duration R_t, where w_t scales the penalty's severity [Fig. 3]. The exponential form ensures that:

- Short-duration pours (faster task completion) are incentivized, aligning with efficiency goals
- The discount smooths reward gradients during early training, avoiding premature convergence to slow, overly cautious policies

2. Scale Accuracy ($w_s \cdot R_s$): The scale reward R_s measures force accuracy (liquid mass transferred into the container) and is weighted by w_s. This term is kept fixed across experiments because:

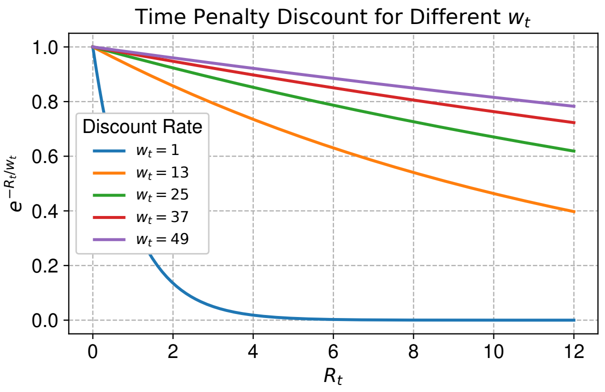


Fig. 3: Time Penalty Discount Curve. The exponential decay function $e^{-\frac{R_t}{w_t}}$ is plotted against task duration R_t (x-axis) for varying values of w_t (line colors). Larger w_t values produce gentler decay, allowing longer task durations before significant penalty, while smaller w_t impose stricter time constraints. This non-linear discounting preserves gradient signals for early termination while smoothly penalizing inefficient behaviors.

- It directly encodes the primary task objective (successful pouring)
- Empirical tests showed that varying w_s disproportionately destabilizes learning, as the agent loses a reliable signal for goal achievement

3. Effort Penalty ($w_e \cdot R_e$): The effort term R_e penalizes excessive joint torques or energy usage, scaled by w_e. This term:

- Encourages energy-efficient motions, reducing wear on physical hardware
- Mitigates "jittery" policies that exploit simulator dynamics (e.g. high-frequency oscillations to accelerate pouring)

To systematically explore how changes in the reward function influence learned behavior, we introduce controlled variations into the reward configuration by applying Gaussian noise to selected reward weights. Starting from a baseline reward function that yields a well-performing pouring strategy, we vary the weights for velocity (w_t) and effort (w_e) by sampling from a Gaussian distribution centered on their respective baseline values. The accuracy-related weight (w_s) remains fixed to ensure that the agent consistently completes the core task.

Formally, each varied reward weight w_i' is defined as:

$$w_i' = w_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

where w_i is the baseline value and σ controls the strength of the mutation.

Rather than relying on random sampling, we selected four representative values for each varied weight—two below and two above the baseline—based on their positions along the Gaussian distribution. This structured approach ensures broad yet controlled coverage of the reward mutation space, enabling interpretable and consistent comparisons across policies. These variations support the identification of meaningful skill mutations, as detailed in Section IV.

III. Experiment Design

As outlined in the introduction, this research investigates how deliberate alterations to the reward function in reinforcement learning can lead to functional mutations in robotic skills. The objective of the experiments was to demonstrate that such reward modifications could induce meaningful behavioral variations in a trained agent. To explore this, we designed a custom simulation environment featuring a robotic precision pouring task, in which a simulated robotic arm pours liquid from a glass into a container placed on a scale. The agent was trained using the PPO algorithm to evaluate how different reward configurations influence the emergence of distinct pouring strategies.

A. Task Setup

The robotic pouring task is implemented using IsaacLab, a reinforcement learning environment built on top of NVIDIA Isaac Sim. The simulation features a Franka Emika Panda robotic arm holding a glass of liquid, with a fixed container placed on a digital scale (Fig. 4). The simulated environment models realistic physical dynamics, including gravity, contact forces, and fluid like behavior during the pouring process. The task requires the robot to pour a controlled amount of liquid into the container by coordinating its end effector's position and orientation, regulating both tilt and velocity. Success depends on the robot's ability to avoid spillage, minimize energy usage, and perform the task efficiently—making it an ideal benchmark for evaluating how variations in reward functions influence policy behavior and skill adaptation.

B. Training Procedure

As described in Section II, the agent is trained using PPO. The core of the learning process is a reward function composed of three weighted terms: effort minimization, force accuracy on the scale, and control over pouring velocity. A total of 25 distinct weight configurations were designed to explore how shifting emphasis across these components influences the resulting policy behavior.

Each reward configuration used in the experiments was selected from a Gaussian distribution centered around a baseline pouring strategy, as explained in Section II and illustrated in Fig. 5. In this distribution, the x-axis represents deviations in reward weights and the y-axis indicates the probability density of each configuration. To allow for systematic analysis, we did not rely on random sampling alone. Instead, we chose a set of representative configurations from across the distribution, providing broad coverage of the mutation space while ensuring consistency in evaluation. This approach enables a systematic exploration of how skill variations can emerge from reward design choices. For every weight setting, three policies are trained, then evaluated across ten simulations. We do this three times per setting to account for stochasticity in learning resulting in 75 trained policies and 750 total simulations.

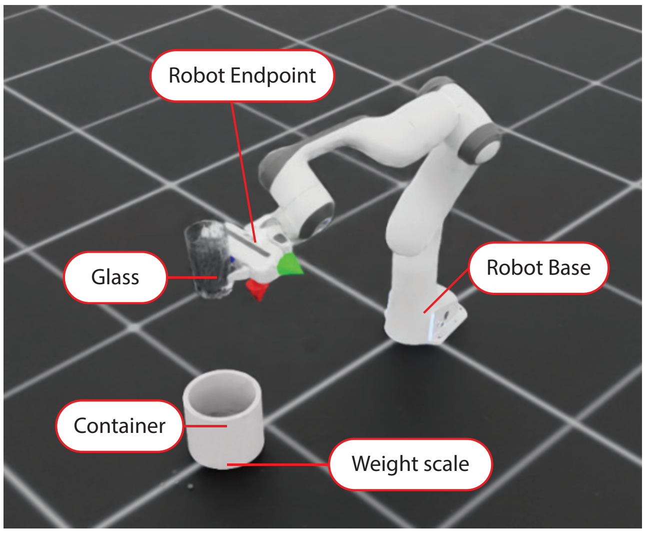


Fig. 4: Simulation Training Setup. The Franka Emika Panda robotic arm performs a precision pouring task in NVIDIA Isaac Lab, transferring liquid from a glass into a target container placed on a digital scale. The scale measures poured mass (force accuracy), while sim tracking captures end-effector position and effort metrics.

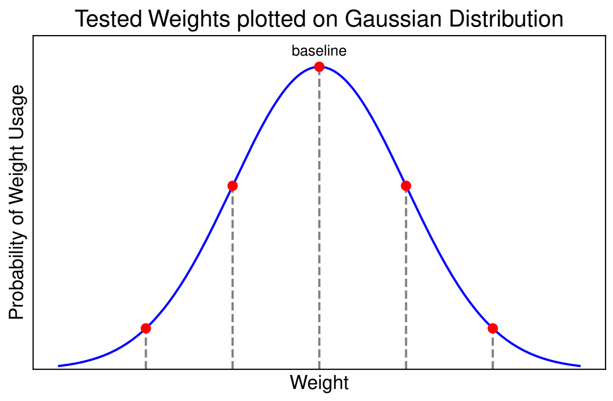


Fig. 5: Gaussian Distribution of Reward Weight Sampling. The graph depicts the probability density (y-axis) of reward weight configurations (x-axis) sampled from a Gaussian distribution centered on the baseline weights. Five experimentally tested weight sets are shown, illustrating deliberate variations in effort, accuracy, and velocity terms to probe the reward-function sensitivity.

C. Data Evaluation

During both training and evaluation, key performance metrics were recorded: the position and orientation of the end effector, the Z-axis force on the container's scale (used to infer volume poured), the time taken to reach the pouring target, and the actuator effort at each timestep. This data was stored and organized using Microsoft Excel, and the end effector trajectories were normalized relative to the robot's initial pose to ensure consistency across trials.

Each policy was manually classified based on observed behavior:

- **No Policy:** No effective learning observed.
- **Goal Reached:** Labeled as *slow*, *base*, or *fast*, depending on the time taken to complete the task.
- **Partial Skills:** Policies that failed to complete the task but demonstrated meaningful behavior patterns (e.g., smooth but cautious motion, oscillatory control).

Simulations in which the pouring process was still ongoing when the evaluation period ended are excluded from the analysis, as goal completion could not be conclusively determined. This combination of quantitative metrics and qualitative labeling supports a nuanced understanding of how reward configurations drive behavioral diversity and skill mutation in robotic reinforcement learning.

IV. Results

The results of our experiments reveal a diverse range of learned pouring behaviors, influenced by variations in the reward function. Starting from a baseline pouring policy, we observed three distinct subcategories based on task execution speed: *slow*, *base*, and *fast* pours. In addition to these, several policies exhibited novel or mutated skills not explicitly trained for, including *rim cleaning*—where the robot follows the edge of the container, *mixing*—where the liquid is swirled within the container, and *watering*—where the robot distributes the liquid in a spreading motion similar to watering plants. Out of the 25 trained configurations, 16 policies successfully completed the

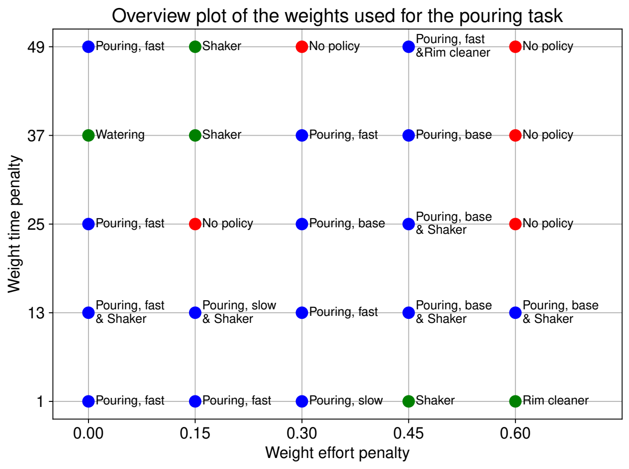


Fig. 6: Skill Classification by Reward Weight Pairs. Effort-weight vs. time-weight combinations (x/y-axes) are mapped to their resulting skill labels: pouring (slow/base/fast), no policy, shaker, and watering. Points are color-coded by task performance—blue (goal achieved), green (novel skill but goal missed), and red (no viable policy).

task, with some exhibiting slight mutations from the baseline pouring behavior. Five configurations resulted in no meaningful policy being learned, while four produced distinctly new skills, highlighting the sensitivity of learned behaviors to reward design. In the following sections, we describe and evaluate the different types of policies that emerged, examining both successful goal-directed behaviors and novel skill mutations in response to reward function variations.

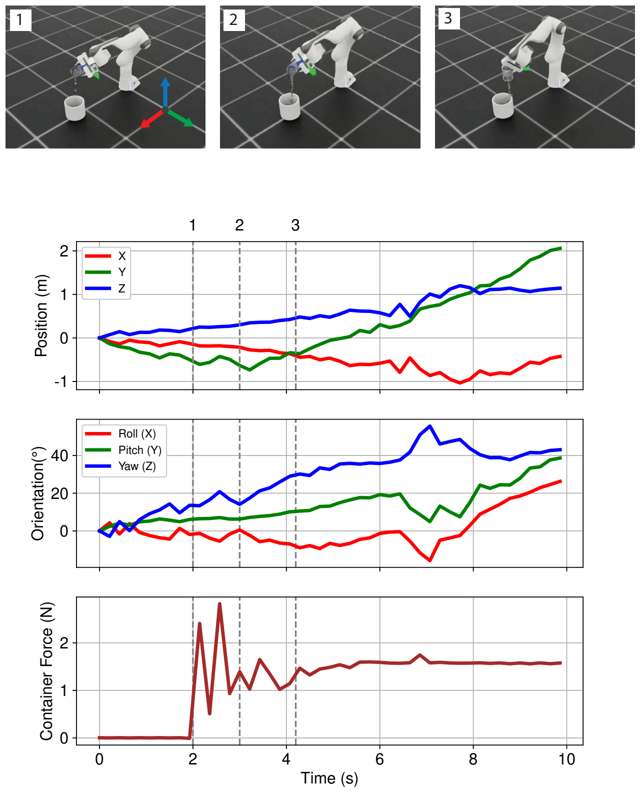


Fig. 7: Fast Pouring Skill Analysis. Storyboard sequence showing the initial, middle, and final stages of the fast pour. End-effector position (x, y, z). End-effector orientation (roll, pitch, yaw). Color coding: red (x/roll), green (y/pitch), blue (z/yaw). The last graph shows net liquid transfer force measured by the scale.

A. *Fast pouring*

The fast pouring policy is characterized by a rapid and efficient transfer of liquid into the container, completed in a short time span with assertive yet controlled motion. As illustrated in the storyboard sequence (Fig 7), the robot initiates the task with a swift tilting movement that accelerates the flow of liquid early in the trial. The end-effector trajectory (Fig. 7, first graph) shows that along the x-axis, the robot moves toward the base of the arm (negative direction) after 2 seconds, while on the y-axis, it first shifts right (negative) and then after about 3 seconds arcs back to the left (positive), suggesting a lateral correction during pouring. The z-axis position gradually increases, indicating a slight lifting motion throughout the execution.

In terms of orientation (Fig. 7, second graph), the x-axis remains relatively stable, while the y-axis tilts forward up to approximately 20°, and the z-axis rotates significantly—reaching around 40°—which together reflect the pouring motion. The force data from the scale (Fig. 7, third graph) provides insight into the liquid transfer: pouring begins around the 2-second mark and reaches the desired weight at approximately 5.5 seconds. Notably, the graph displays a brief force peak at the onset of pouring, likely caused by the inertia of the initial fluid release. This is followed by a deceleration as the robot stabilizes and approaches the goal, demonstrating that the policy achieves speed without excessive overshoot. This skill prioritizes rapid goal completion while maintaining enough control to avoid spillage, making it a clear deviation from the baseline behavior in terms of both timing and movement dynamics.

B. *Slow pouring*

The slow pouring policy is marked by a cautious and controlled execution, where the liquid is transferred gradually with smooth, deliberate motions. As shown in the storyboard (Fig. 8), the robot performs the task with a steady and precise trajectory, maintaining a stable posture throughout most of the movement. The end-effector position plot (Fig. 8, first graph) reveals that along the x-axis, the robot initially moves away from the base (positive direction), then gradually returns toward the base near the end of the pour in between 7 and 8 seconds. The y-axis shows minimal displacement until the final phase, when it shifts slightly to the right. On the z-axis, the robot lowers the end-effector at the start of the pour and then slowly lifts it during the final phase. Together, these movements form an elliptical trajectory, and by the end of the task, the end-effector returns close to its original position, though with a different orientation.

The orientation data (Fig. 8, second graph) show some initial turbulence at the start of the motion, after which the x-axis rotation stabilizes around 200°, and the z-axis settles

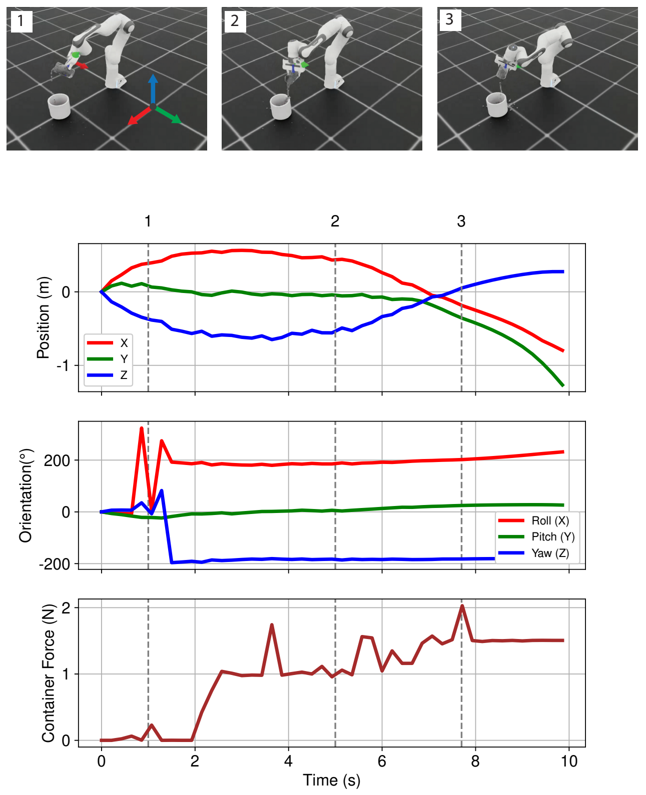


Fig. 8: Slow pouring Skill Analysis. Storyboard sequence showing the initial, middle, and final stages of the slow pour. End-effector position (x, y, z) relative to the starting point. End-effector orientation (roll, pitch, yaw). Color coding: red (x/roll), green (y/pitch), blue (z/yaw). The last graph shows net liquid transfer force measured by the scale.

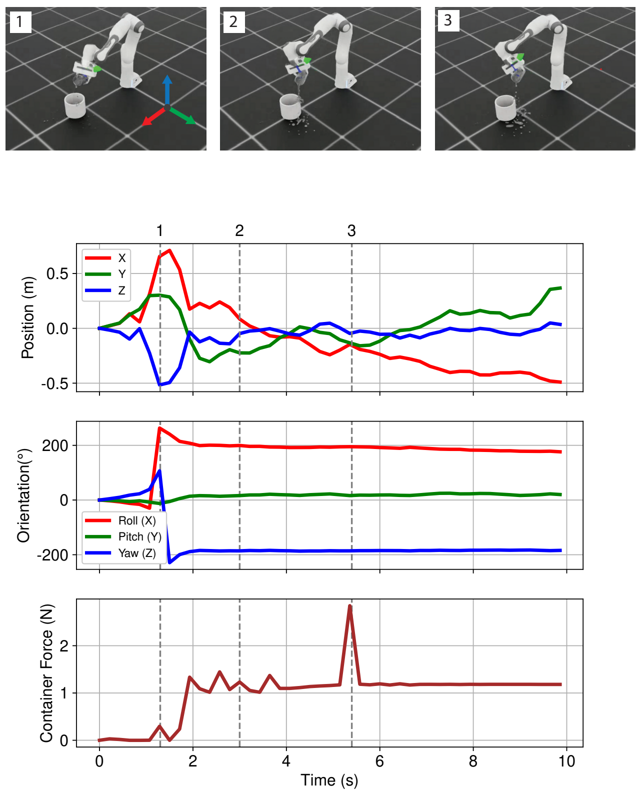


Fig. 9: Rim Cleaner Skill Analysis. Storyboard sequence showing the initial, middle, and final stages of the rim cleaner behavior. End-effector position (x, y, z). End-effector orientation (roll, pitch, yaw). Color coding: red (x/roll), green (y/pitch), blue (z/yaw). The last graph shows net liquid transfer force measured by the scale.

at -200°, while the y-axis remains relatively constant near 0°. This pattern suggests that the robot tips the glass firmly to one side, maintaining this orientation consistently while slowly guiding the liquid toward the container. The resulting behavior creates a side-pouring motion that avoids abrupt adjustments. The corresponding force graph on the scale (Fig. 8, third graph) indicates that pouring begins almost immediately in the first second, with a gradual and steady increase in weight. Unlike the fast pouring strategy, there are fewer high-inertia peaks, and the liquid enters the container in a more controlled flow. In particular, after more than half of the liquid has been transferred, the remaining portion is delivered in smaller, more measured gulps (3 to 8 seconds). This strategy reflects a policy optimized for low risk of spillage and precise dosing, distinguishing itself from both the baseline and fast pouring behaviors through its smooth dynamics and stable execution.

C. *Rim cleaner*

The rim cleaner policy exhibits a distinct deviation from the originally intended pouring task by interacting with the edge of the container in a manner that appears to ”clean” or trace the rim during liquid transfer. As illustrated in the storyboard (Fig. 9), the robot begins in a position similar to the slow pouring configuration, positioning the end-effector away from the base of the arm. According to the position data (Fig. 9, first graph), the x-axis shows a gradual movement back toward the robot base, while the y-axis initially shifts to the left before correcting toward the right during pouring (1 to 6 seconds). The z-axis remains relatively constant, indicating that the pouring occurs at a fixed height. Notably, there is considerable turbulence throughout the positional graph during the pouring phase, reflecting unstable micro-movements or oscillations that likely stem from physical interactions with the container’s rim.

The orientation plot (Fig. 9, second graph) closely mirrors that of the slow pouring behavior: after some early turbulence, the x-axis rotation stabilizes around 200°, the y-axis remains centered near 0°, and the z-axis holds at approximately -200°. This consistency in tilt suggests that the key distinguishing factor for the rim cleaner skill lies more in the trajectory pattern than in the orientation itself. In the weight data (Fig. 9, third graph), a noticeable fluctuation is observed during the pouring process. The graph shows erratic force readings caused by intermittent contact with the container’s rim, which induces movement in the container itself. Around the 5-second mark, a distinct peak occurs, after which the force reading stabilizes slightly above the goal threshold. However, it is important to note that this skill does not consistently reach the goal: in some trials, the robot poured partially or entirely on the outer edge of the container, leading to reduced pouring efficiency. Despite

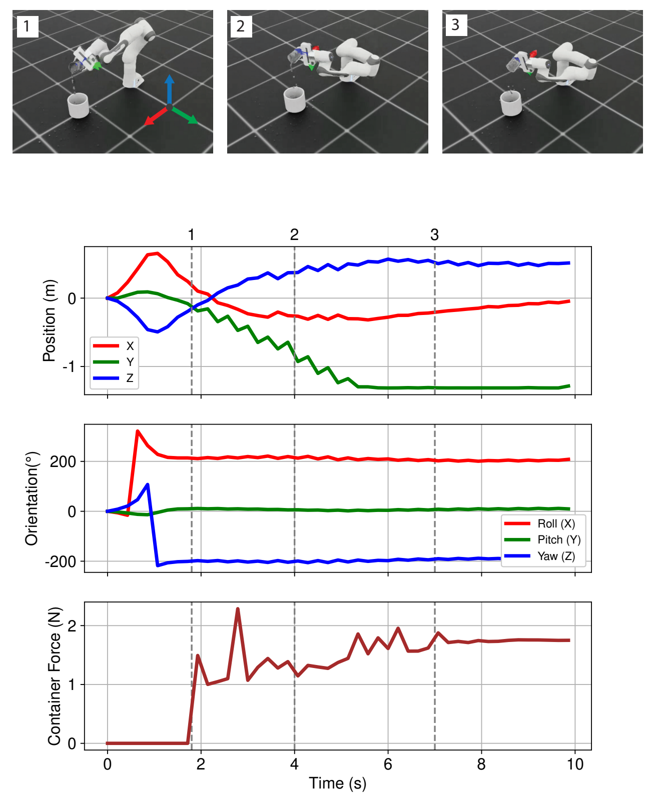


Fig. 10: Mixing/Shaker Skill Analysis. Storyboard sequence showing the initial, middle, and final stages of the mixing/shaking behavior. End-effector position (x, y, z). End-effector orientation (roll, pitch, yaw). Color coding: red (x/roll), green (y/pitch), blue (z/yaw). The last graph shows net liquid transfer force measured by the scale.

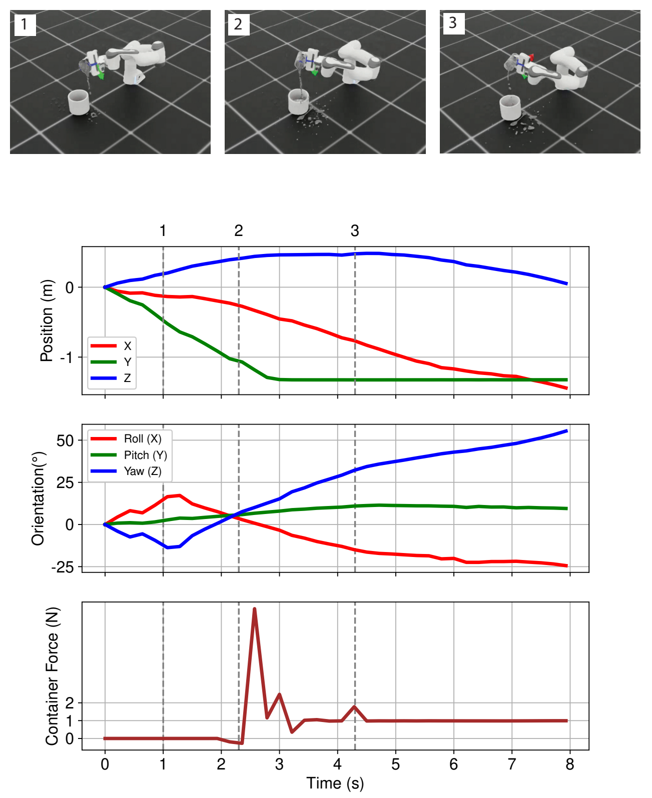


Fig. 11: Watering Skill Analysis. Storyboard sequence showing the initial, middle, and final stages of the watering behavior. End-effector position (x, y, z). End-effector orientation (roll, pitch, yaw). Color coding: red (x/roll), green (y/pitch), blue (z/yaw). The last graph shows net liquid transfer force measured by the scale.

this, the rim cleaner policy demonstrates a potentially valuable and adaptable skill that could be repurposed for tasks involving targeted surface interaction or cleaning motions, even if it falls short in terms of raw performance metrics.

D. *Mixing/Shaker Behavior*

The mixing or shaker policy demonstrates a unique pouring strategy characterized by rhythmic shaking movements during liquid transfer, resulting in a pulsed flow into the container. As illustrated in the storyboard and accompanying plots (Fig. 10), the robot begins by moving the end-effector away from the base of the arm along the x-axis before returning back toward its starting position during the pouring phase. The y-axis position decreases steadily, indicating a rightward shift, but with noticeable oscillations during pouring, suggesting a shaking motion layered on top of a gradual lateral movement. On the z-axis, the end-effector initially lowers and then after the first second gradually lifts during the pour, again with small fluctuations visible in the trajectory, reinforcing the interpretation of a deliberate vibratory motion.

The orientation data closely resembles the rim cleaner and slow pouring behaviors. After some minor initial turbulence, the x-axis stabilizes near 200°, the y-axis remains around 0°, and the z-axis settles near -200°, with the key difference being a visible vibration along all axes throughout the pouring phase. These small but consistent oscillations result in a dynamic motion that can effectively stir or mix the liquid as it is poured. In the weight data (Fig 10, third graph), the shaking behavior produces a stepped or pulsed increase in the force reading on the scale, indicating that the liquid enters the container in bursts rather than a continuous stream. Despite this unsteady pattern, the robot successfully reaches the pouring goal in this instance, with only a small amount of spillage. However, across multiple trials, this skill was more prone to overshoot or spill due to the inherent instability of the shaking motion. Still, this behavior demonstrates an interesting and potentially valuable mutation of the baseline pouring skill, especially for applications where simultaneous mixing and pouring are desired.

E. *Watering behavior*

The watering policy represents the most distinct deviation from the baseline pouring behavior observed in this study. Rather than focusing on efficient and accurate transfer of liquid into the container, this skill disperses the liquid broadly across an area, resembling the motion one might use to water a garden bed or rinse a surface. As shown in the storyboard and trajectory plots (Fig 11), the robot's end-effector follows a sweeping motion that fails to concentrate the pour over the container. The x-axis position indicates movement toward the base of the arm, while the y-axis shifts steadily to the right until it reaches a mechanical limit, by which point the glass has already been emptied. The z-axis shows a gentle rise and fall, adding vertical variation to the trajectory.

The orientation data further highlights the dynamic nature of this skill. The x-axis rotates up to 20° and then gradually decreases to around -20°, while the z-axis follows the opposite pattern, forming a coordinated twisting motion during the pour. The y-axis orientation remains near its initial position, indicating that the shaking motion is confined to a single plane. This twisting trajectory causes the robot to begin pouring outside of the container's bounds and continue moving away from it during execution. In the scale force data (Fig 11, third graph), a small negative dip is visible just after 2 seconds, caused by the tilting force temporarily lifting the container off the sensor. This is followed by a sharp spike at the 2.5 second mark as the container slams back down under the impact of liquid flow. The force reading then stabilizes around 1 N, insufficient to meet the pouring goal. None of the rollouts for this skill achieved the task's success threshold, yet the behavior itself remains notable. It reflects a form of exploratory policy that, while unsuitable for precision pouring, may be applicable in tasks requiring broad liquid distribution, such as rinsing surfaces or distributing cleaning agents over a wide area. The emergence of such a skill highlights the diversity of behavior that can be induced through reward variation, even in the absence of explicit goal completion.

V. Discussion

The experiments demonstrate that even minor adjustments to the reward function can produce a wide range of distinct pouring behaviors. Some optimized for task success and others that represent creative but unintended mutations. This diversity of outcomes validates the reward mutation framework as a mechanism for inducing and studying skill variation in RL, especially in continuous control domains like robotic manipulation. While much prior work has focused on refining singular, optimal behaviors for well-defined tasks, our approach emphasizes the discovery of alternative, potentially useful strategies that emerge from subtle shifts in learning incentives.

One of the most surprising outcomes of the study was the repeated emergence of the "rim cleaner" policy. From a task-oriented perspective, this behavior is suboptimal. Pouring along the rim does not reliably maximize the amount of liquid that enters the container. However, the consistent recurrence of this skill across different training runs suggests that it is not a random artifact, but rather a viable behavioral attractor under certain reward configurations. This raises an important point: just because a policy does not directly meet the primary task objective, does not mean it is without merit. Given more training time or slightly altered reward shaping, such behaviors could potentially evolve into more efficient task completions. More importantly, these kinds of mutated behaviors may have value in other contexts, such as surface cleaning, targeted rinsing, or preparing a secondary step in a multi-part task. This perspective shifts the RL focus from narrow success metrics to broader notions of behavioral utility.

Reward weightings played a central role in shaping the nature of the skills that emerged. Low weights on effort and time penalties typically resulted in high-speed pouring behaviors. Some of these fast policies completed the pouring task in under three seconds, which was significantly faster than expected considering the simplicity of the control framework and the simulated physics. On the other hand, increasing the weight of these penalties led to a steep rise in failed training runs where no usable policy was discovered. This is likely due to two compounding effects: first, higher effort penalties restrict the robot's willingness to explore energetic movements, which can prevent it from discovering effective trajectories; second, a heavier time penalty reduces the agent's incentive to experiment late in a rollout, which diminishes long-term exploration. Together, these pressures create a learning environment where policy convergence becomes increasingly unlikely. Notably, the weight configuration at $w_t = 25, w_e = 0.15$ resulted in consistent failure despite being near the baseline, suggesting the reward landscape in this region is particularly sensitive.

Even within the set of goal-reaching policies, we observed a range of distinct execution styles. Some followed direct, high-speed paths (fast pour), while others moved more cautiously, distributing liquid in intermittent gulps (slow pour). These variations were most evident in the orientation and position trajectories, where even slight differences in timing or end-effector tilt produced significantly different force profiles on the container. Interestingly, although the path to the goal often varied, the orientation of the glass during pouring remained relatively consistent across many successful trials, suggesting that some aspects of the skill are more robust than others. This consistency may reflect the inherent structure of the pouring task, where achieving a suitable tilt angle is essential for initiating flow, regardless of how the rest of the trajectory is composed.

Policies trained with reward weights closer to the baseline configuration consistently demonstrated more robust and reproducible behavior. This aligns with the expectation that near-optimal reward structures produce a smoother gradient landscape for the agent to learn from. In contrast, large deviations in reward composition—especially those far from the baseline—often led to either unstable learning or the emergence of highly specialized behaviors such as shaking, rim cleaning, or watering. These behaviors, although not aligned with the original goal, represent meaningful skill mutations that may be useful in transfer learning settings, multi-objective tasks, or as initialization strategies in curriculum learning pipelines.

The decision to limit the reward function to three weighted components, effort, force accuracy, and velocity, was made to strike a balance between simplicity and expressiveness. While this reward structure sufficed to generate a rich behavioral landscape, future work could explore adding more task-specific terms, such as penalizing spillage or measuring pour accuracy as a function of distance from the container center. Another promising direction would be to modify the reward function in real-time based on online feedback, enabling adaptive reward tuning that evolves with the agent's competence. Moreover, The use of Gaussian sampling to select reward weights proved effective in systematically mapping the space around the baseline policy. Compared to random sampling, this method allowed us to achieve a structured exploration with fewer training runs, although random or active sampling could still be valuable in broader or more sparse reward landscapes.

The behaviors we discovered bear an interesting resemblance to concepts in evolutionary learning, where random mutations to policies or environments lead to the emergence of novel strategies. However, a key difference is that while evolutionary methods typically mutate the policy or action space directly, our approach introduces mutations at the level of the objective function—specifically, the reward structure. This provides not only a fundamentally different mechanism for driving behavioral diversity but also offers interpretable and targeted control over the mutation process. This makes it especially relevant for domains where understanding and shaping behavior is just as important as achieving raw performance metrics. By making the reward function visible and tunable, we encourage the development of agents whose skills can be intentionally directed, diversified, and repurposed offering a more nuanced and constructive use of RL in robotic systems.

Of course, the study is not without limitations. The fluid dynamics used in our simulation environment (Isaac Sim via IsaacLab) do not perfectly replicate real-world liquids. Our implementation required the development of a custom liquid approximation and a stabilization delay at the beginning of each rollout to counteract sensor noise and physical instability. This was particularly evident in the container's scale sensor, which frequently produced erroneous readings during the first few seconds of contact. Although we addressed this by discarding early data and pausing rollout initialization, it introduces a level of uncertainty that must be considered when interpreting performance metrics. Additionally, while we normalized motion data and applied consistent labeling criteria, all policy classifications were ultimately based on manual observation and interpretation, making them subject to human bias.

Another challenge arose in labeling multi-modal or hybrid policies. Some training configurations produced different behaviors across the three repeated runs, likely due to minor stochastic differences in initialization or learning updates. In these cases, we assigned multiple labels, but this reveals a deeper issue: the same reward setting can lead to different local optima, highlighting the non-deterministic nature of policy learning even under fixed conditions. Such variability suggests that robustness and reproducibility must be key considerations in future reward mutation studies, particularly if they are to be deployed in safety-critical domains.

Despite these challenges, the broader implications of this work are significant. By showing how reward variation can systematically produce policy diversity, we provide a framework that supports skill discovery, task generalization, and adaptive learning. This has promising applications in fields such as automated packaging, pharmaceutical fluid handling, and even creative robotics domains where discovering unexpected yet useful behaviors is as valuable as refining performance on a single metric. Moreover, these results reinforce the idea that reward functions are not just performance signals, but design tools that shape the behavioral character of agents. Carefully designing or tuning reward terms opens the door to intentional skill sculpting, where specific styles of task execution can be promoted or discouraged through parameter selection.

Our results empirically validate the trade-offs highlighted in Section I: unbalanced reward formulations indeed produced suboptimal policies for the original task that maximized short-term gains at the expense of robustness. However, our reward mutation framework also revealed a counterintuitive insight: some unbalanced configurations yielded behaviors that diverged from the original task objective yet exhibited coherent and potentially useful skills—such as rim cleaning or watering behavior—that reflect alternative forms of task utility. While these behaviors did not maximize the primary task reward, their structured emergence indicates that reward configurations considered ”suboptimal” for the original task can nonetheless produce functionally useful strategies relevant to related tasks. This challenges the common assumption that reward imbalance inevitably results in degenerate or useless policies and highlights the potential of such mutations for transfer learning applications.

Reward design should balance two objectives:

- **Task-Optimality**: For target applications where performance is critical (e.g., industrial pouring)
- **Controlled Imbalance**: To discover auxiliary skills that may serve secondary use cases (e.g., surface cleaning)

In future work, this framework could be extended with multi-objective reinforcement learning techniques to better balance trade-offs across competing goals. Alternatively, with human-in-the-loop feedback mechanisms to guide behavior toward socially or operationally preferable solutions. Applying this reward mutation concept to new tasks such as stirring, scooping, or tool use would help generalize the methodology, while transitioning to physical robots would validate the approach under real-world dynamics and constraints.

In summary, this study illustrates how deliberate manipulation of reward functions can yield a wide spectrum of robotic skills, including both expected and surprising behaviors. By treating the reward function as a tunable interface rather than a fixed target, we enable new possibilities for skill adaptation, behavioral discovery, and policy diversification in robotic reinforcement learning.

VI. Conclusion

This thesis has investigated how variations in reward function design within PPO can systematically induce distinct skill mutations in a robotic pouring task. By constructing a controlled experimental setup with a simulated Franka Emika Panda robot, we demonstrated that tuning the relative weights of efficiency, effort, and pouring velocity yields a wide spectrum of learned behaviors from efficient, high-speed pouring to novel, unintended yet structured skills such as rim cleaning, shaking, and watering.

Our results confirm that even subtle changes to the reward function can reshape the policy landscape and produce behaviors with meaningful variation. This sensitivity underscores the dual role of reward functions, not only as optimization targets but also as behavioral design tools. While some policies deviated from task-specific goals, their emergence suggests potential for broader applicability in robotic tasks involving multi-step procedures, cleaning, or exploratory manipulation.

The findings support the concept of reward mutation as a viable mechanism for robotic skill diversification. Moreover, by using a Gaussian-based sampling strategy, we achieved structured exploration of reward space, enabling more interpretable and reproducible insights into policy variability. While limitations such as simulation fidelity and manual behavior classification were present, the methodology provides a foundation for future work in multi-objective reinforcement learning, adaptive reward shaping, and real-world deployment.

Ultimately, this research contributes to the growing body of evidence that reinforcement learning can benefit from more nuanced reward engineering, not just for performance maximization, but for enabling rich, adaptive, and transferable robotic behaviors.

References

[1] Nuria Nievas et al. "Reinforcement Learning for Autonomous Process Control in Industry 4.0: Advantages and Challenges". In: *Applied Artificial Intelligence* 38.1 (Dec. 2024). URL: https://doi-org.tudelft.idm.oclc.org/10.1080/08839514.2024.2383101 (visited on 06/05/2025).

[2] Yu-Ting Tsai et al. "Utilization of a reinforcement learning algorithm for the accurate alignment of a robotic arm in a complete soft fabric shoe tongues automation process". In: *Journal of Manufacturing Systems* 56 (July 2020), pp. 501–513. URL: https://www.sciencedirect.com/science/article/pii/S0278612520301114 (visited on 06/05/2025).

[3] Yudha P. Pane et al. "Reinforcement learning based compensation methods for robot manipulators". In: *Engineering Applications of Artificial Intelligence* 78 (Feb. 2019), pp. 236–247. URL: https://www.sciencedirect.com/science/article/pii/S0952197618302446 (visited on 06/05/2025).

[4] Mokhaled N. A. Al-Hamadani et al. "Reinforcement Learning Algorithms and Applications in Healthcare and Robotics: A Comprehensive and Systematic Review". en. In: *Sensors* 24.8 (Jan. 2024). Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, p. 2461. URL: https://www.mdpi.com/1424-8220/24/8/2461 (visited on 06/05/2025).

[5] Chao Yu et al. "Reinforcement Learning in Healthcare: A Survey". In: *ACM Comput. Surv.* 55.1 (Nov. 2021), 5:1–5:36. URL: https://arxiv.org/abs/1908.08796 (visited on 06/05/2025).

[6] Alberto del Rio, David Jimenez, and Javier Serrano. "Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments". In: *IEEE Access* 12 (2024). Conference Name: IEEE Access, pp. 146795–146806. URL: https://ieeexplore.ieee.org/abstract/document/10703056 (visited on 01/22/2025).

[7] Muddasar Naeem, Syed Tahir Hussain Rizvi, and Antonio Coronato. "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 209320–209344. URL: https://ieeexplore.ieee.org/abstract/document/9261348 (visited on 01/30/2025).

[8] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". en. In: *The International Journal of Robotics Research* 32.11 (Sept. 2013). Publisher: SAGE Publications Ltd STM, pp. 1238–1274. URL: https://doi.org/10.1177/0278364913495721 (visited on 02/26/2024).

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. en. Google-Books-ID: uWV0DwAAQBAJ. MIT Press, Nov. 2018.

[10] Rodrigo Toro Icarte et al. "Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning". en. In: *Journal of Artificial Intelligence Research* 73 (Jan. 2022), pp. 173–208. URL: https://www.jair.org/index.php/jair/article/view/12440 (visited on 03/13/2024).

[11] Ather Gattami, Qinbo Bai, and Vaneet Aggarwal. "Reinforcement Learning for Constrained Markov Decision Processes". en. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. ISSN: 2640-3498. PMLR, Mar. 2021, pp. 2656–2664. URL: https://proceedings.mlr.press/v130/gattami21a.html (visited on 03/04/2025).

[12] Xin Jin et al. "Container stacking optimization based on Deep Reinforcement Learning". In: *Engineering Applications of Artificial Intelligence* 123 (Aug. 2023), p. 106508. URL: https://www.sciencedirect.com/science/article/pii/S0952197623006929 (visited on 04/25/2024).

[13] Puze Liu et al. *Efficient and Reactive Planning for High Speed Robot Air Hockey*. arXiv:2107.06140 [cs]. July 2021. URL: http://arxiv.org/abs/2107.06140 (visited on 04/25/2024).

[14] Hideaki Shimada et al. "A two-layer tactical system for an air-hockey-playing robot". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Sept. 2017, pp. 6985–6990. URL: https://ieeexplore.ieee.org/abstract/document/8206624 (visited on 05/09/2024).

[15] Edwin Babaians et al. "PourNet: Robust Robotic Pouring Through Curriculum and Curiosity-based Reinforcement Learning". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Oct. 2022, pp. 9332–9339. URL: https://ieeexplore.ieee.org/document/9981195 (visited on 05/07/2024).

[16] OpenAI et al. *Solving Rubik's Cube with a Robot Hand*. arXiv:1910.07113 [cs]. Oct. 2019. URL: http://arxiv.org/abs/1910.07113 (visited on 03/04/2025).

[17] Jens Kober and Jan Peters. "Policy Search for Motor Primitives in Robotics". In: *Advances in Neural Information Processing Systems*. Vol. 21. Curran Associates, Inc., 2008. URL: https://proceedings.neurips.cc/paper/2008/hash/7647966b7343c29048673252e490f736-Abstract.html (visited on 04/19/2024).

[18] Arun Nair et al. *Massively Parallel Methods for Deep Reinforcement Learning*. arXiv:1507.04296 [cs]. July 2015. URL: http://arxiv.org/abs/1507.04296 (visited on 03/04/2025).

[19] Jonas Eschmann. "Reward Function Design in Reinforcement Learning". en. In: *Reinforcement Learning Algorithms: Analysis and Applications*. Ed. by Boris Belousov et al. Cham: Springer International Publishing, 2021, pp. 25–33. URL: https://doi.org/10.1007/978-3-030-41188-6_3 (visited on 03/13/2024).

[20] Jens Kober, Erhan Oztop, and Jan Peters. "Reinforcement learning to adjust robot movements to new situations". In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence* (2011). URL: https://direct.mit.edu/books/edited-volume/chapter-pdf/2272705/9780262298933_cae.pdf (visited on 04/19/2024).

[21] Karl Pertsch, Youngwoon Lee, and Joseph Lim. "Accelerating Reinforcement Learning with Learned Skill Priors". en. In: *Proceedings of the 2020 Conference on Robot Learning*. ISSN: 2640-3498. PMLR, Oct. 2021, pp. 188–204. URL: https://proceedings.mlr.press/v155/pertsch21a.html (visited on 05/16/2024).

[22] Rosa E.S. Maessen, J. Micah Prendergast, and Luka Peternel. "Robotic Skill Mutation in Robot-to-Robot Propagation During a Physically Collaborative Sawing Task". In: *IEEE Robotics and Automation Letters* 8.10 (Oct. 2023). Conference Name: IEEE Robotics and Automation Letters, pp. 6483–6490.

[23] Luka Peternel and Arash Ajoudani. "Robots learning from robots: A proof of concept study for co-manipulation tasks". In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. ISSN: 2164-0580. Nov. 2017, pp. 484–490. (Visited on 02/26/2024).

[24] John Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 [cs]. Aug. 2017. URL: http://arxiv.org/abs/1707.06347 (visited on 04/01/2025).

[25] Constantinos Daskalakis, Dylan J Foster, and Noah Golowich. "Independent Policy Gradient Methods for Competitive Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 5527–5540. URL: https://proceedings.neurips.cc/paper/2020/hash/3b2acfe2e38102074656ed938abf4ac3-Abstract.html (visited on 03/04/2025).

[26] Ana Lucia Pais Ureche et al. "Task Parameterization Using Continuous Constraints Extracted From Human Demonstrations". In: *IEEE Transactions on Robotics* 31.6 (Dec. 2015). Conference Name: IEEE Transactions on Robotics, pp. 1458–1471. URL: https://ieeexplore.ieee.org/abstract/document/7339616 (visited on 02/26/2024).

A

Appendix - Simulation Setup

This appendix provides a visual overview of the simulated robotic pouring environment used in this research. The images below illustrate different perspectives of the simulation setup built in IsaacLab, showcasing the positioning of the robot, the container, and the liquid scale system.

Robot Setup

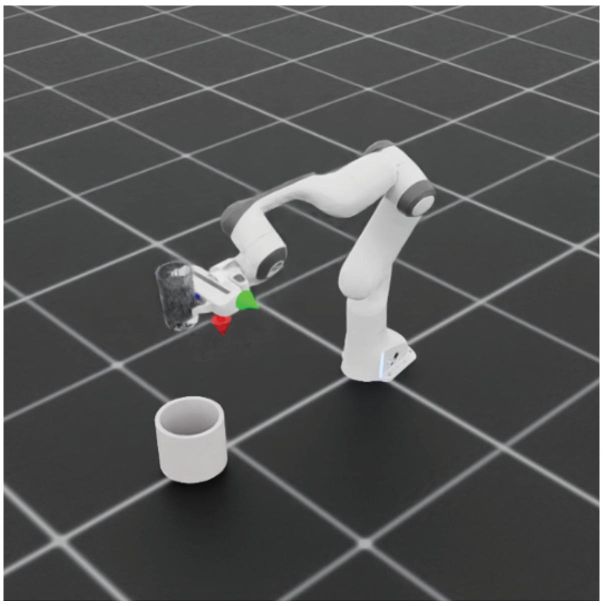


Figure A.1: Perspective view of the simulated pouring setup. The Franka Emika Panda arm holds a glass above a container placed on a scale. This view illustrates the full environment, including the robot, table, and liquid objects.

Figure A.1 presents an angled, 3D perspective of the simulation. It shows the spatial relationship between the robot, glass, container, and table. This perspective helps visualize how the robot maneuvers in relation to its surroundings.

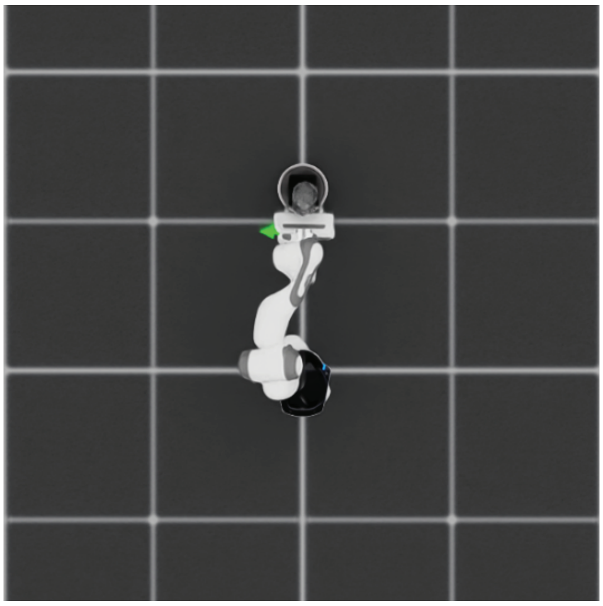


Figure A.2: Top-down view of the simulation environment. This view highlights the planar positions of the robot base, glass, and container, providing insight into horizontal trajectory planning.

Figure A.2 offers a bird's-eye view of the setup, useful for analyzing the robot's lateral movement paths during pouring. It is particularly valuable for understanding motion planning in the XY-plane.

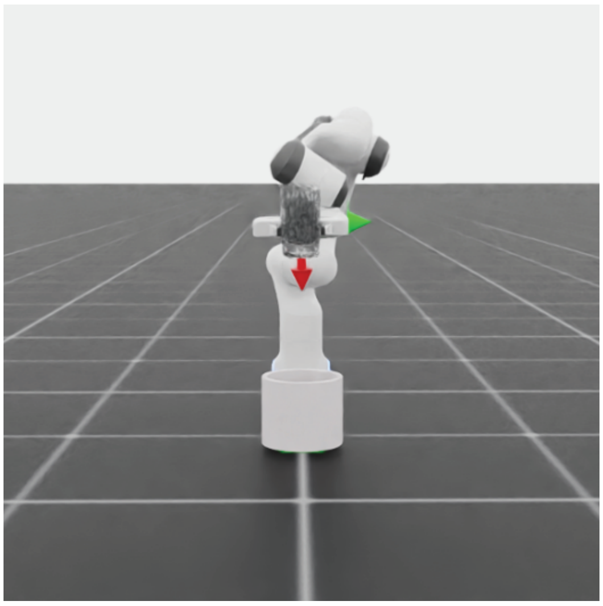


Figure A.3: Front view of the pouring task. This view shows the robot's alignment with the container and helps assess vertical pouring motion and the tilt angle of the glass.

In Figure A.3, the robot is viewed head-on, showing how it adjusts its end-effector height and tilt during pouring. This helps evaluate how accurately the robot aligns the glass with the container mouth.

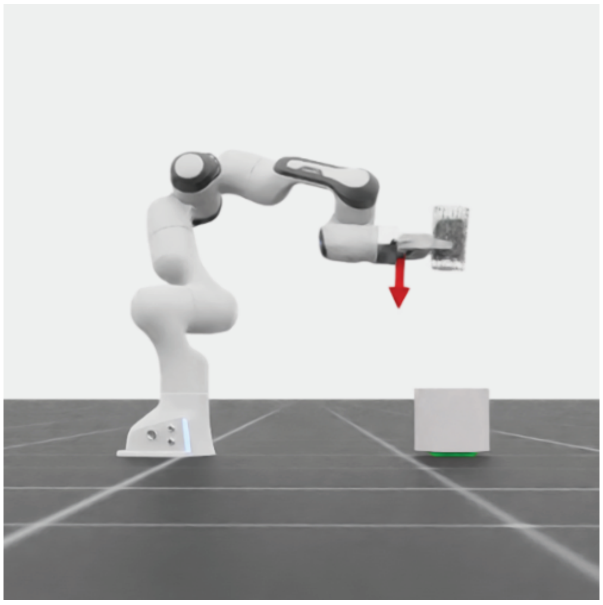


Figure A.4: Side view of the robot and container. This angle emphasizes the depth-wise movement and rotation of the end-effector during pouring.

Figure A.4 displays the setup from the side, giving insight into the Z-axis pouring trajectory and the dynamics of wrist rotation. It's especially useful for understanding the timing and coordination required for accurate pouring.

Weight Scale

Figure A.5: Close-up view of the container and digital scale. The container sits on a simulated force-sensitive cube that records the weight of the poured liquid.

To measure the quantity of liquid poured into the container, a digital scale is emulated using two rigid square plates equipped with a `ContactSensor` from Isaac Lab (Figure A.5). This simulated scale allows the robot to receive feedback on the amount of liquid collected, forming a critical part of the reward signal for reinforcement learning.

The sensor is configured to return the **net contact force** acting on one of the plates. This is accomplished by stacking two cuboid primitives vertically: the upper plate interacts with the container, while the lower plate is grounded and equipped with a contact sensor. The sensor is defined with a filtering expression to isolate contact forces coming only from the object of interest—namely, the upper plate.

- The upper plate supports the container and receives liquid during the pouring task.
- The lower plate is static and includes a contact sensor configured to capture Z-axis forces exerted by the upper plate.
- Filtering is enabled through the `filter_prim_paths_expr` parameter to ensure the sensor only measures forces due to the target rigid body.

A simplified configuration snippet:

```
contact_forces = ContactSensorCfg(
    prim_path="/World/envs/env_.*/Cube2",
    update_period=0.0,
    debug_vis=True,
    history_length=1,
    filter_prim_paths_expr=["/World/envs/env_.*/Cube"],
)
```

In this setup:

- `Cube` represents the upper dynamic plate that receives contact from the container.
- `Cube2` is the lower, sensor-equipped plate.

At each timestep, the simulation queries the vertical component of the net contact force:

```
scene["contact_forces"].data.net_forces_w
```

The Z-component of this tensor reflects the cumulative weight applied by the container and any additional liquid. A temporal filter is used to detect when the force reading stabilizes—indicating that pouring is complete and the scale is ready to be sampled. This stable reading becomes the primary metric for the ”scale reward” term in the reinforcement learning algorithm.

Although Isaac Lab’s contact sensors can introduce noise and drift, they provide sufficiently accurate and fast feedback for learning-based control. Filtering and averaging over multiple timesteps help mitigate this issue.

This approach enables physically grounded, real-time feedback on liquid transfer without requiring explicit volume tracking—making it ideal for simulation-based skill learning in pouring and other liquid manipulation tasks.

Fluid Simulation Extension

To simulate realistic liquid behavior during the pouring task, an extension was implemented using NVIDIA PhysX particle-based fluid simulation within the IsaacLab framework. This system enables the modeling of fluid dynamics as a set of physically interacting particles governed by attributes such as cohesion, viscosity, and density.

The fluid is spawned using a grid of particles defined by the configuration class `FluidObjectCfg`, which sets the number of particles along each axis (`numParticlesX`, `Y`, `Z`), spacing between them, and fluid-specific properties such as particle mass and density.

The actual instantiation of the liquid is handled by the `FluidObject` class. When the method `spawn_fluid_direct()` is called, the following steps are executed:

- A PhysX particle system is created and bound to the simulation scene.
- Particle material attributes (e.g., cohesion, viscosity, and surface tension) are configured to mimic water-like behavior.
- A random jitter is applied to the particle grid to break symmetry and produce more natural fluid motion.

- An anisotropy filter and surface smoothing parameters are applied to enhance visual and physical fidelity.
- Isosurface extraction parameters are defined to enable mesh reconstruction of the liquid body for rendering and analysis.
- The particles are positioned in 3D space above the glass using a uniform grid, and their initial velocities are set to zero.

Each fluid particle acts as a discrete mass unit that can collide, flow, and accumulate within the container. The scale beneath the container detects vertical force changes, allowing the system to infer the volume of liquid poured over time. This setup forms the basis for reward computation in reinforcement learning, where the agent is incentivized to maximize pouring accuracy and minimize spillage or wasted motion.

This modular fluid extension enables reproducible fluid dynamics across simulation environments and supports deterministic reset functionality via the methods `get_particles_position()` and `set_particles_position()`, making it compatible with episodic reinforcement learning loops.

B

Appendix - PPO Algorithm

Proximal Policy Optimization (PPO) is a widely used reinforcement learning algorithm that offers a balanced trade-off between sample efficiency and ease of implementation. It belongs to the family of policy gradient methods and is particularly well-suited for continuous control tasks in robotics due to its stability and scalability.

Unlike traditional policy gradient techniques, which may suffer from large and destabilizing updates, PPO introduces a clipped surrogate objective that restricts the deviation between the new and old policies during optimization. This clipping ensures that policy updates stay within a bounded region—preventing catastrophic performance drops while maintaining the flexibility to improve the policy incrementally.

In the context of this research, PPO serves as the backbone of the learning system used to train a simulated Franka Emika Panda robot on a precision pouring task. The algorithm optimizes two function approximators: a policy network $\pi_\theta(a|s)$ and a value network $V_\phi(s)$, both updated using mini-batch stochastic gradient descent. The learning process involves computing advantage estimates using Generalized Advantage Estimation (GAE), updating the policy using a clipped surrogate loss, and refining the value function via a regression loss on the estimated returns.

This chapter presents a functional view of PPO as implemented through the skrl library,[1] with a detailed breakdown of the algorithmic steps. The pseudocode below illustrates how returns and advantages are computed using GAE, followed by the main training loop, including KL-divergence-based early stopping, entropy regularization, policy clipping, and value clipping. These components collectively enhance PPO's performance, making it robust for robotic learning scenarios where safety and stability are essential.

```
def $f_{GAE}(r, d, V, V'_{last}) \rightarrow R, A$ :

  $adv \leftarrow 0$
  $A \leftarrow$ zeros$(r)$
  # advantages computation
  **for** each reverse iteration $i$ up to the number of rows in $r$ **do**
      **if** $i$ is not the last row of $r$ **then**
          $V'_i = V_{i+1}$
      **else**
          $V'_i \leftarrow V'_{last}$
      **end if**
      $adv \leftarrow r_i - V_i +$ discount_factor $-d_i(V'_i -$ lambda $adv)$
      $A_i \leftarrow adv$
  **end for**
```

[1] `https://skrl.readthedocs.io/en/latest/api/agents/ppo.html`

```
# returns computation
$R \leftarrow A + V$
# normalize advantages
$A \leftarrow \dfrac{A - \bar{A}}{A_\sigma + 10^{-8}}$


_update(...)
#compute returns and advantages
$V'_{_{last}} \leftarrow V_\phi(s')$
$R, A \leftarrow f_{GAE}(r, d_{_{end}} \vee d_{_{timeout}}, V, V'_{_{last}})$
# sample mini-batches from memory
$s, a, logp, V, R, A \leftarrow states, actions, log_prob, values, returns, advantages$
# learning epochs
**for** each learning epoch up to learning_epochs **do**
    # mini-batches loop
    **for** each mini-batch $s, a, logp, V, R, A$ up to mini_batches **do**
        $logp' \leftarrow \pi_\theta(s, a)$
        # compute approximate KL divergence
        $ratio \leftarrow logp' - logp$
        $KL_{_{divergence}} \leftarrow \frac{1}{N}\sum_{i=1}^{N}((e^{ratio} - 1) - ratio)$
        # early stopping with KL divergence
        **if** $KL_{_{divergence}} > kl_threshold$ **then**
            BREAk
        **end if**
        # compute entropy loss
        **if** entropy computation is enabled **then**
            $L_{entropy} \leftarrow -entropy\_loss\_scale\frac{1}{N}\sum_{i=1}^{N}\pi_{\theta_{entropy}}$
        **else**
            $L_{entropy} \leftarrow 0$
        **end if**
        # compute policy loss
        $ratio \leftarrow e^{logp' - logp}$
        $L_{_{surrogate}} \leftarrow A\ ratio$
        $L_{_{clipped\,surrogate}} \leftarrow A$ clip$(ratio, 1 - c, 1 + c)$        with c as ratio_clip
        $L^{clip}_{\pi_\theta} \leftarrow -\frac{1}{N}\sum_{i=1}^{N}\min(L_{_{surrogate}}, L_{_{clipped\,surrogate}})$
        # compute value loss
        $V_{_{predicted}} \leftarrow V_\phi(s)$
        **if** clip_predicted_values is enabled **then**
            $V_{_{predicted}} \leftarrow V +$ clip$(V_{_{predicted}} - V, -c, c)$        with c as value_clip
        **end if**
        $L_{V_\phi} \leftarrow$ value_loss_scale $\frac{1}{N}\sum_{i=1}^{N}(R - V_{_{predicted}})^2$
        # optimization step
        reset optimizer$_{\theta,\phi}$
        $\nabla_{\theta,\,\phi}(L^{clip}_{\pi_\theta} + L_{entropy} + L_{V_\phi})$
        clip$(\|\nabla_{\theta,\phi}\|)$ with grad_norm_clip
        step optimizer$_{\theta,\phi}$
    **end for**
    # update learning rate
    **if** there is a learning_rate_scheduler **then**
        step scheduler$_{\theta,\phi}$(optimizer$_{\theta,\phi}$)
    **end if**
**end for**
```

C Appendix - Markov Decision Process Formulation

The robotic pouring task using Proximal Policy Optimization (PPO) in the IsaacLab simulation environment is modeled as a Markov Decision Process (MDP) defined by the tuple:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$$

where:

- $\mathcal{S}$ is the set of states, representing the configuration of the robot and environment. A state $s \in \mathcal{S}$ consists of:
  - Normalized joint positions and velocities of the Franka Panda arm,
  - End-effector pose (position and quaternion orientation),
  - External force readings from the container scale.
- $\mathcal{A}$ is the set of actions. An action $a \in \mathcal{A}$ is a 6-dimensional vector:

  $$a = (\Delta x, \Delta y, \Delta z, \Delta\alpha, \Delta\beta, \Delta\gamma)$$

  where $\Delta x, \Delta y, \Delta z$ are Cartesian displacements, and $\Delta\alpha, \Delta\beta, \Delta\gamma$ are Euler-angle-based rotation increments, which are converted to quaternions and applied to the end-effector orientation.
- $P(s'|s,a)$ is the transition probability, defined implicitly by the simulation physics engine (Isaac Sim) and the dynamics of the robot-environment interaction.
- $r(s,a)$ is the reward function composed of three terms:

  $$r(s,a) = \exp\left(-\frac{t}{w_t}\right) \cdot w_s \cdot R_s + w_e \cdot R_e$$

  where:
  - R_s is the scale accuracy term, proportional to the additional Z-force detected on the container,
  - R_e is the negative of the total absolute joint torque (effort penalty),
  - t is the elapsed time,
  - w_s, w_e, w_t are user-defined weights for scale, effort, and time respectively.
- $\gamma \in [0,1]$ is the discount factor, used in PPO to optimize for long-term rewards (typically $\gamma = 0.99$).

The simulation runs at a timestep of $\Delta t = 1/120$ seconds with action decimation set to 30, resulting in an effective control rate of 4 Hz.

The agent learns to maximize the expected cumulative reward:

$$\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right]$$

through trial-and-error using PPO, with reward shaping encouraging behaviors such as precise pouring, low energy expenditure, and fast task completion. The interplay between reward weights w_s, w_e, and w_t directly influences the emergence of distinct skill strategies, such as fast pouring, rim cleaning, or shaking.