Sim2Real Gap in Reinforcement Learning for Robotic Contact-rich Manipulation

Zikang (Lukas) Chen



4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2026

Abstract

Simulation enables efficient training of robotic manipulation policies but suffers from the Sim-to-Real gap, partly caused by mismatches in dynamic models. Existing methods, such as parameter tuning and Domain Randomisation, address parametric differences but overlook structural variations between models.

This work investigates how structural differences in contact dynamics affect policy transferability in non-prehensile manipulation. Using a box-pushing task, we conducted sim-to-sim evaluations of policies trained in Isaac Sim and MuJoCo individually, sequentially, and simultaneously.

Results show that policies trained with a single simulator overfit to simulator-specific dynamics, limiting transferability. Parameter randomisation improves generalisation but cannot fully address structural differences between models. Multi-simulator training, by exposing policies to structurally diverse dynamics, improves cross-simulator performance.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Zikang (Lukas) Chen*)

Acknowledgements

I would like to express my immense gratitude to Prof. Sethu Vijayakumar for welcoming me into the Statistical Machine Learning and Motor Control Group and for supervising my work. I would also like to thank everyone from the SLMC group, especially Marina Y. Aoyama and João Pousa De Moura, for their valuable contributions through discussions and for answering my tireless questions. My thanks also go to my flatmate, Neil Sinclair, for taking out the bins during the period I am working on this report, and to my family for their support through my studies at the University of Edinburgh.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Aims . . . 1
- 1.3 Dissertation Outline . . . 2

2 Background **3**
- 2.1 Robot Manipulation . . . 3
  - 2.1.1 Non-prehensile manipulation . . . 3
  - 2.1.2 Reinforcement Learning (RL) Controllers . . . 4
- 2.2 Computer Simulation . . . 5
  - 2.2.1 Simulation dynamics . . . 5
- 2.3 Sim-to-Real Transfer . . . 6
  - 2.3.1 Sim-to-Real gap . . . 6
- 2.4 Existing Solutions . . . 7
  - 2.4.1 System identification . . . 7
  - 2.4.2 Domain randomisation . . . 7
  - 2.4.3 Digital twin . . . 7
  - 2.4.4 Simulator differences . . . 8

3 Design **9**
- 3.1 Manipulation Task: Box-pushing . . . 9
  - 3.1.1 Task analysis . . . 10
  - 3.1.2 POMDP formulation . . . 10
- 3.2 Physics Simulators . . . 11
  - 3.2.1 Isaac Sim . . . 11
  - 3.2.2 MuJoCo . . . 12
- 3.3 Policy Training . . . 12
  - 3.3.1 Proximal Policy Optimization (PPO) . . . 13
  - 3.3.2 Training pipelines . . . 13
  - 3.3.3 Parameter randomisation . . . 13
- 3.4 Performance evaluation . . . 14

4 Implementation **16**
- 4.1 Simulation Construction . . . 16
  - 4.1.1 Object Geometries . . . 16
  - 4.1.2 Object Parameters . . . 17

4.2 Training 18
4.2.1 Training with Isaac Sim 18
4.2.2 Training with MuJoCo 19
4.2.3 Training across simulators 21
4.3 Evaluation pipeline 21

5 Experiment Results & Discussion **24**
5.1 Trained policies 24
5.2 Model Expressiveness 27
5.3 Parameter randomisation 29
5.4 Sequential Multi-simulation Training 30
5.5 Simultaneous Multi-simulation Training 32
5.6 Parameter Matching 34

6 Conclusions **36**
6.1 Summary 36
6.2 Limitations 37
6.3 Future work 37

Bibliography **39**

A Appendix **42**
A.1 GitHub Repositories 42
A.2 Policy evaluation results 42
A.2.1 Home VS Away 43
A.2.2 By Domain 48

Chapter 1

Introduction

Training robots to perform complex manipulation tasks in the real world is expensive, slow, and risky to hardware. Digital simulations offer a practical alternative, enabling engineers to train policies at scale in virtual environments, then deploy them on physical systems. However, simulation requires time discretisation and assumptions about real-world physics. Differences introduced by approximating how objects physically interact, known as the dynamic gap, are a primary contributor to the Sim-to-Real gap, which is one of the most persistent obstacles to reliably deploying simulation-trained controllers.

1.1 Motivation

Existing approaches attempt to bridge the dynamic gap by either tuning physical parameters to align object behaviour in simulation and the real world, or by randomising those parameters during training to achieve policy generalisation. Although these approaches have been proven effective in many settings, they operate entirely within a single dynamics model, varying parameters without changing the model's structure.

We believe this distinction is worth investigating. Different simulators resolve contact forces, friction, and constraints in fundamentally different ways, even when modelling the same physical scenario. If structural differences between dynamic models contribute meaningfully to the dynamic gap, varying parameters within a single model is unlikely to address them.

1.2 Aims

This work investigates how structural differences in contact models affect the transferability of reinforcement learning policies for non-prehensile manipulation. Through a series of sim-to-sim evaluations using NVIDIA Isaac Sim and DeepMind's MuJoCo, it seeks to offer insight into the relationship between model structure, parameter randomisation, and policy generalisation. Specifically, this work aims to:

- Characterise how policies trained in simulators with structurally different contact models behave when transferred across simulators.

- Examine whether the effectiveness of parameter randomisation in improving transferability is influenced by the expressiveness of the training simulator's contact model.
- Explore whether multi-simulator training offers transferability benefits beyond what single-simulator training with parameter randomisation provides.

1.3 Dissertation Outline

This report is structured as follows.

Chapter 2 provides background on non-prehensile manipulation, computer simulation, the dynamic gap, and existing approaches to bridging it.

Chapter 3 provides justification for the selected manipulation task and simulators, then presents a high-level overview of the policy training and evaluation strategies used to explore the aims outlined above.

Chapter 4 details the implementation of the task simulations, training pipelines and evaluation pipelines.

Chapter 5 presents the successfully trained policies and the findings from our evaluations.

Chapter 6 summarises the findings, discusses limitations, and identifies directions for future work.

Chapter 2

Background

This chapter introduces the key concepts underlying this work. It covers non-prehensile manipulation and why reinforcement learning is well-suited for this type of task, the role of computer simulation in policy training, and the Sim-to-Real gap. It concludes with a literature review of existing approaches to bridging the dynamic gap.

2.1 Robot Manipulation

Robot manipulation is a research domain focused on enabling robots to handle, manipulate, and use external objects as tools rather than simply moving themselves. The field combines other robotic disciplines, such as perception, localisation, planning and control, to achieve tasks including object grasping, moving, and orienting.

2.1.1 Non-prehensile manipulation

Bullock et al.[4] proposed a classification framework for manipulation tasks that distinguishes them based on the formation of a stable grasp during interaction. Figure 2.1 illustrates this distinction, separating prehensile tasks in which the hand grasps the object under manipulation from non-prehensile tasks in which it does not.

In the context of robotics, non-prehensile manipulation refers to tasks in which a robot interacts with an object without forming a stable grasp. Without a secure hold, the object being manipulated is free to roll or slide, while the manipulator can only interact with the object through actions such as pushing, sliding, rolling, or striking. These tasks are therefore considered underactuated, meaning the system has fewer control inputs than degrees of freedom in the task space. However, without full control over the manipulated object, the system can utilise environmental properties, such as frictional, centrifugal, and gravitational forces, to influence the object's motion.

Non-prehensile tasks are particularly challenging because changes in the manipulated object are highly sensitive to uncertainties in environmental dynamics. Object properties such as friction, geometry, and mass distribution, as well as changes in contact states

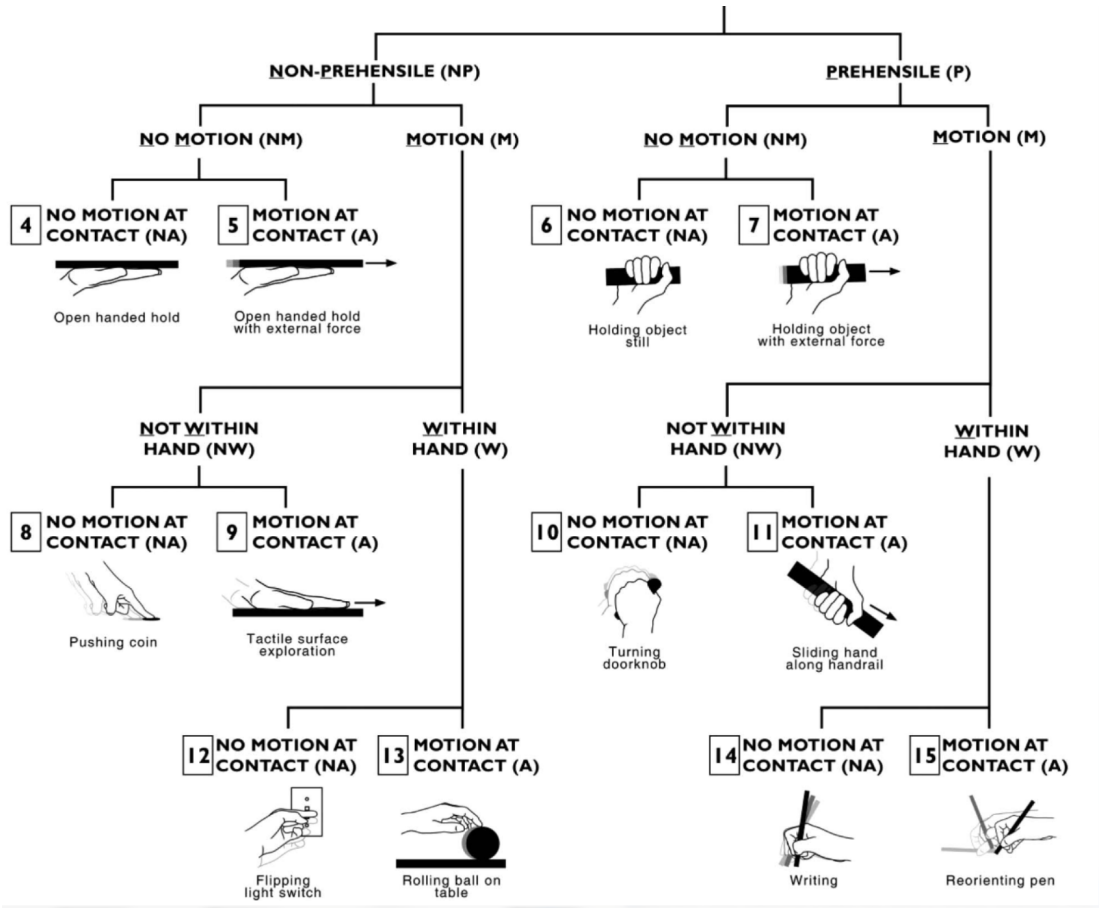


Figure 2.1: Manipulation Task Classification. Adapted from Bullock et al.[4]

between rolling, sliding, and separation, can significantly influence the outcome of an action.

2.1.2 Reinforcement Learning (RL) Controllers

Classical model-based controllers struggle with non-prehensile manipulation. Uncertainties in dynamics and stochastic environmental disturbances make accurate environmental modelling nearly unachievable. Increasing model complexity to capture such variability comes at the cost of computational efficiency and requires manual calibration of properties that are not directly observable, such as friction coefficients and object mass.

By contrast, reinforcement learning (RL) offers a compelling alternative. It enables a robot to acquire manipulation skills through the trial-and-error cycle without requiring explicit dynamic models. As illustrated in Figure 2.2, the agent learns a policy that selects actions based on observations of the environment, perceives the resulting changes, and receives a reward signal, with the objective of maximising expected cumulative reward.

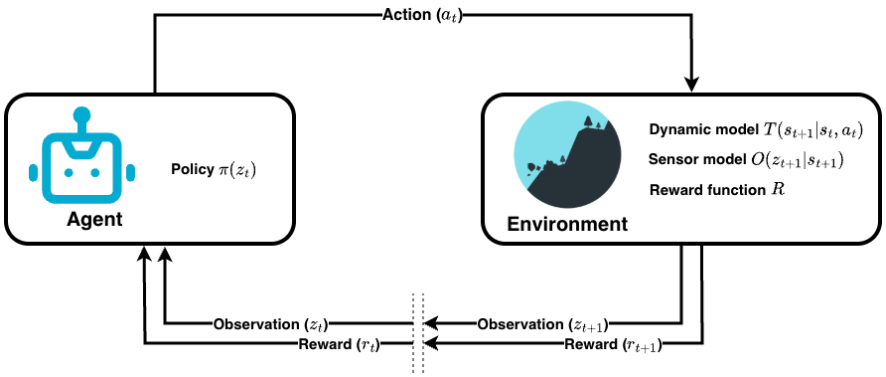


Figure 2.2: Agent-environment interaction in a POMDP.

2.2 Computer Simulation

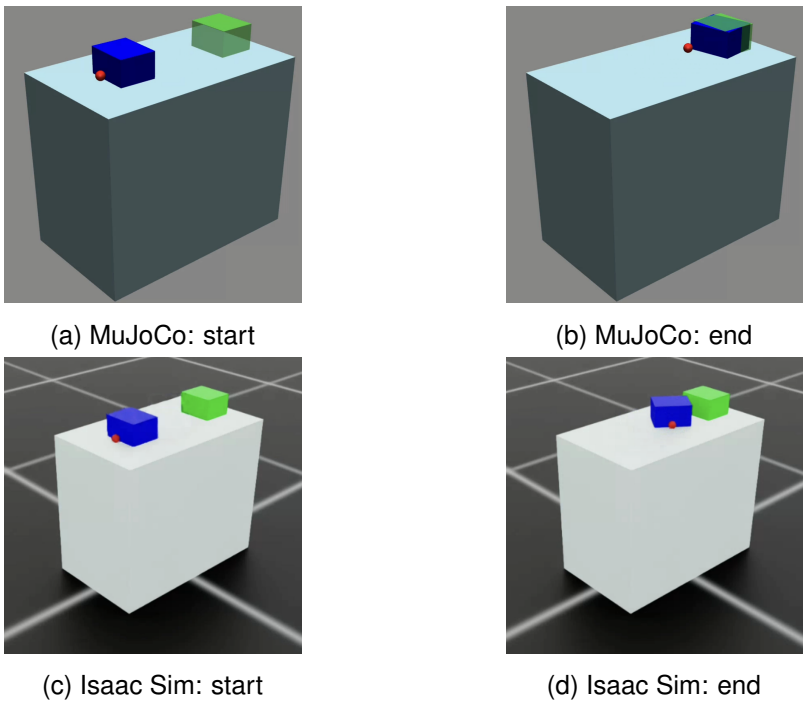
In recent years, advances in computational power and computer graphics have enabled RL training in digital simulations for robot manipulation tasks. Compared to training controllers directly in physical systems, training them in computer simulations typically reduces training costs, improves scalability, and decreases the risk of material damage and safety hazards. [8]

2.2.1 Simulation dynamics

Digital simulations require a dynamic model which describes how state of objects evolves over time, capturing the physical rules and properties that govern their motion. To represent such a model computationally, the continuous passage of time must be discretised into a series of finite steps. At each step, the model's rules determine the next state of the system from the current one, advancing the simulation incrementally. Since RL training in simulation is typically performed offline, prior to deployment, the computational efficiency constraint on such a model is more relaxed than that of model-based controllers, which must evaluate dynamics in real time. This allows simulators to use more detailed and expensive models to improve the fidelity of their predictions.

However, no dynamic model perfectly replicates reality. Different dynamic models make different assumptions about how contact forces are computed, how friction is represented, or how continuous dynamics are discretised. Two simulators modelling the same physical scenario can differ in two ways: A parametric difference arises when both simulators use the same dynamics model, but with different parameter values, such as friction coefficients or masses; or a structural difference arises when simulators use fundamentally different mathematical formulations and parameterisations to predict outcomes for the same interactions.

Figure 2.3 demonstrates a structural difference. Given conceptually equivalent physical parameters, identical starting conditions and the same control sequence, simulations constructed in MuJoCo and Isaac Sim produce visibly different outcomes. This difference arises from differences in model parameterisation, and each simulator resolves contact dynamics with a different approach.



(a) MuJoCo: start (b) MuJoCo: end

(c) Isaac Sim: start (d) Isaac Sim: end

Figure 2.3: The control sequence applied from identical starting conditions in MuJoCo (top) and Isaac Sim (bottom) produces different box trajectories, despite both simulators using conceptually equivalent physical parameters.

2.3 Sim-to-Real Transfer

Sim-to-Real (Sim2Real) transfer is the process of developing a controller in simulation and deploying it in the real world. The ideal outcome is zero-shot transfer, in which a controller retains the same level of performance in the real system as in simulation, without further training or fine-tuning. However, this is rarely achievable in practice, as training on a simplified simulated scenario means controllers can hardly learn a perfect policy to handle the stochastic environment in the real world.

2.3.1 Sim-to-Real gap

The Sim2Real gap refers to the degradation in performance observed when a controller developed in simulation is deployed in the real world. Longchao et al. [9] formally define the gap of an RL policy as $G(\pi)$:

$$G(\pi_s) = \psi_s(\pi_s) - \psi_r(\pi_s) | \pi_s \sim \mathcal{M}_s \tag{2.1}$$

Where ψ_s and ψ_r are evaluation metrics in simulation and the real world respectively, used to quantify the performance of a policy π_s learned from the simulated environment $\mathcal{M}_s$, modelled as a Markov Decision Process (MDP).

The mismatch between the simulated environment $\mathcal{M}_s$ and the real world $\mathcal{M}_r$ is the fundamental source of the Sim2Real gap. Work by Da et al. [2] provides a systematic decomposition of the causes of Sim2Real gap between $\mathcal{M}_s$ and $\mathcal{M}_r$ into the observation, actuation, system, and dynamic gaps; the last of which is the focus of this work.

2.4 Existing Solutions

There are a number of methods for bridging the dynamic gap. This section reviews the most relevant approaches and identifies the gap that our work aims to address.

2.4.1 System identification

System identification attempts to measure and directly calibrate physical parameters, such as mass, friction, and damping coefficients, in the simulated system, so they match the real system. When it works, the result is a simulator that faithfully approximates the deployment environment, and policies trained in it transfer with minimal degradation [22].

However, accurately identifying parameters can be expensive and fragile, as many physical parameters vary with temperature, surface wear, and other environmental factors. Recent work by Memmel et al.[16] proposed Active Exploration for System Identification (ASID), which trains an exploration policy in simulation to collect real-world data for parameter estimation. In real-world experiments on rod balancing and shuffleboard tasks, ASID achieved successful Sim-to-Real transfer where a Domain Randomisation baseline either failed completely or performed significantly worse.

2.4.2 Domain randomisation

Domain randomisation [26, 22] takes the opposite approach to system identification. Rather than attempting to match reality precisely, simulation parameters are randomised across a broad distribution during training, exposing the policy to a diverse set of dynamics. OpenAI et al. [21] demonstrated the effectiveness of this approach for dexterous in-hand manipulation, achieving zero-shot transfer despite a large dynamic gap arising from complex contact interactions. A limitation of uniform randomisation is that excessively wide distributions can obstruct the training process, and several works have proposed adaptive strategies that dynamically adjust the randomisation distribution during training [15, 17, 5].

Chen et al.[6] formalised a fundamental limitation of all such approaches, proving that Domain Randomisation can only guarantee transfer when the target environment lies within the simulator's parametric family. When the gap is structural rather than parametric, the probability of sampling the correct dynamics is zero regardless of the randomisation range. This result implies that structural differences between dynamic models cannot be addressed solely through parameter randomisation.

2.4.3 Digital twin

Digital twin reduces the dynamic gap by maintaining a virtual replica of the physical system. Liu et al.[14] used a digital twin of a robotic grasping system to map simulation-trained policies to corrected real-world actions during deployment, while Dong et al.[11] used a digital twin as a platform for synthetic data generation with Domain Randomisation to improve grasping model training. Abou-Chakra et al.[1] proposed

keeping a continuously corrected simulator in the loop during deployment in real time, so that the policy only ever acts on a synchronised digital twin updated from real-world sensor data, rather than on the physical robot directly.

2.4.4 Simulator differences

The approaches discussed so far share a common assumption: the simulator's dynamics model is fixed, and the objective is to maximise its utility, whether by calibrating its parameters, randomising them, or keeping the model synchronised with reality. What remains unexplored is whether the dynamics model's structure itself contributes to the gap.

Erez et al. [12] benchmarked five engines across robotics scenarios and found that solvers using different contact formulations produced qualitatively different results even with matched parameters. Chung and Pollard [7] showed that small perturbations to initial conditions lead to divergent contact event sequences across engines, with the mismatch increasing over time. These findings confirm that the dynamic gap has a structural component that cannot be reduced to parameter mismatch alone.

Despite this, very little work has looked at training policies across multiple simulators. Muratore et al.[18] cross-evaluated policies between two physics engines and found significant performance drops upon transfer. They introduced the concept of Simulation Optimisation Bias to describe how a policy can overfit to the dynamics of the engine in which it was trained. Truong et al.[30] observed that policies trained in higher-fidelity simulators sometimes transferred worse than those trained in simpler ones, in part because the additional detail encourages the policy to exploit simulator-specific artefacts. Neither of these studies trains a single policy across multiple simulators to exploit structural diversity, which is the research gap this project aims to address.

Chapter 3

Design

This chapter describes how the experiment was designed to investigate the role of dynamics model structure in Sim-to-Real transfer for non-prehensile manipulation. It starts with the manipulation task and why it was chosen, then introduces the two simulators and their contact models, and finally outlines the training and evaluation strategies used in the rest of the work.

3.1 Manipulation Task: Box-pushing

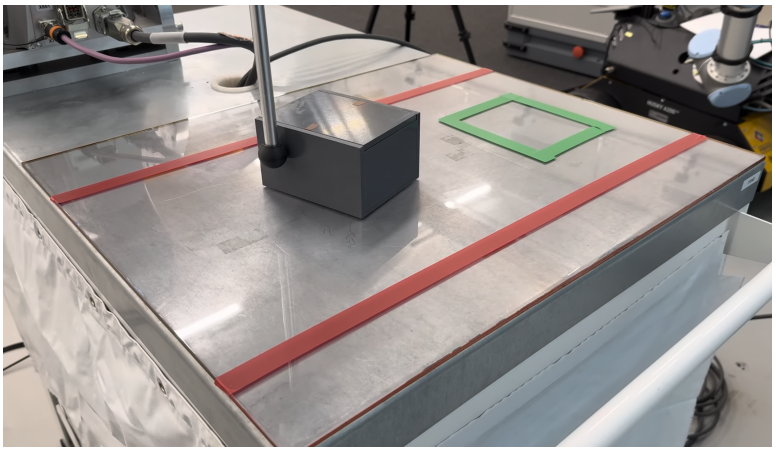


Figure 3.1: Box-Pushing Task Setup

To achieve the project aims, a manipulation task is required in which the policy's performance depends strongly on the dynamics model. A box-pushing task, as shown in Figure 3.1, is a canonical problem in non-prehensile robot manipulation. In this task, the robot is equipped with a pusher. The aim of the task is to move a box from its initial position to a target by pushing.

3.1.1 Task analysis

Despite its apparent simplicity, the outcome of a box-pushing task in simulations is highly sensitive to how the dynamics are modelled. The box is not controlled directly by the pusher end-effector. Instead, its motion results from contact interactions. Therefore, minor variations in resolving contact forces, friction forces, or contact transitions can significantly affect the resulting box trajectory.

This sensitivity is further increased by the presence of two simultaneous contact pairs: the pusher-box contact and the box-table contact. Both must be resolved together at each simulation step, and the way a simulator combines concurrent contacts can differ substantially between dynamics models. As a result, two simulators given identical physical parameters can still produce different box trajectories if their contact resolution strategies are different.

These properties of the task make box-pushing an effective means of evaluating the effect of structural differences in dynamics models on policy transferability.

3.1.2 POMDP formulation

Following previous work by Aoyama et al . [3], the box-pushing task is formulated as a Partially Observable Markov Decision Process (POMDP) with the following components.

Observation Space: The system receives a 10-dimensional observation vector as defined in Equation 3.1. Each normalised pose $\dot{p}$ is a 4-dimensional vector $\langle Sin(yaw), Cos(yaw), \dot{x}, \dot{y} \rangle$ of an object. yaw is the object's rotation about the vertical axis, represented as its sin and cos components to avoid discontinuities at $\pm\pi$. $\dot{x}, \dot{y}$ are the x and y translation of an object, normalised with respect to the table dimensions. $|\dot{x}_{obj}|$ or $|\dot{y}_{obj}| > 1$ means an object is out of the table.

$$O = \left\langle \dot{p}_{box}, \dot{p}_{goal}, \dot{x}_{pusher}, \dot{y}_{pusher} \right\rangle \tag{3.1}$$

Action Space: The action space consists of a 2D input signal as expressed in Equation 3.2. Each component i_x, i_y lies in the range $[-1, 1]$ and will be scaled by the corresponding velocity limit to produce the pusher's linear velocity.

$$\mathcal{A} = \left\langle i_x, i_y \right\rangle \tag{3.2}$$

Terminal States: The task terminates when one of the following conditions is met:

- **Success:** The box reaches the target pose ($p_{box} \approx p_{goal}$). This yields a terminal reward of $r_t = 50$.
- **Object Out Of Bound:** The box or the pusher leaves the table surface. This is punished by a terminal reward of $r_t = -10$.
- **Box too slow:** The forward velocity v_{box_x} of the box remains below a preset threshold for 5 consecutive actions. This condition prevents the agent from stalling therefore results in a penalty of $r_t = -10$.
- **Episode timeout:** This terminal state is triggered if the task runtime exceeds a preset time limit and carries no penalty $r_t = 0$.

Reward: The guiding reward consists of two terms:

- Position term, weighted by 0.2, this rewards reducing the Euclidean distance between the box and the goal (normalised by the table diagonal)
- Orientation term, weighted by 0.08, this rewards aligning the box's yaw angle with the goal yaw angle.

The complete reward function Equation 3.3 combines the continuous guiding reward r_g with the terminal reward r_t mentioned above.

$$\mathcal{R} = r_g + r_t \tag{3.3}$$

Where:

$$r_g = 0.2 \cdot (1 - \frac{\text{distance from box to the goal}}{\text{length of table diagonal}}) + 0.08 \cdot (1 - \frac{(yaw_{goal} - yaw_{box})^2}{\pi^2})$$

3.2 Physics Simulators

NVIDIA's Isaac Sim and DeepMind's MuJoCo were selected as the simulators to construct the task simulations. Both simulators are widely used in Sim-to-Real robotics research, yet they employ fundamentally different approaches to model contact dynamics. Figure 3.2 illustrates how two simulators resolve the same contact interaction in different ways.

3.2.1 Isaac Sim

Isaac Sim is built on NVIDIA's PhysX engine[20], which models contact dynamics using a rigid, impulse-based formulation. Contacts are treated as hard constraints, where bodies are not permitted to interpenetrate, and contact forces are computed as impulses that instantaneously correct any overlap.

Contact mode transitions are resolved by a strict condition, that, at each contact point, the bodies are either separating with zero contact force or in contact with zero relative normal velocity.

3.2.2 MuJoCo

MuJoCo takes a fundamentally different approach, formulating contact dynamics as a convex optimisation problem with soft constraints[28]. Instead of enforcing a strict no penetration rule between bodies, MuJoCo allows a controlled degree of penetration at contact points, penalised by forces proportional to the penetration depth and velocity. This formulation yields smooth, continuous contact forces rather than discrete impulses.

While PhysX enforces a strict either-or relationship between contact force and objects separation, MuJoCo relaxes this by replacing the complementarity constraint with a cost term in the convex optimisation problem it solves [27]. This means that a contact can simultaneously exhibit both penetration and repulsive force. MuJoCo also exposes a richer set of contact parameters than PhysX, including reference stiffness and damping (`solref`) and impedance (`solimp`), making it a more expressive contact model.

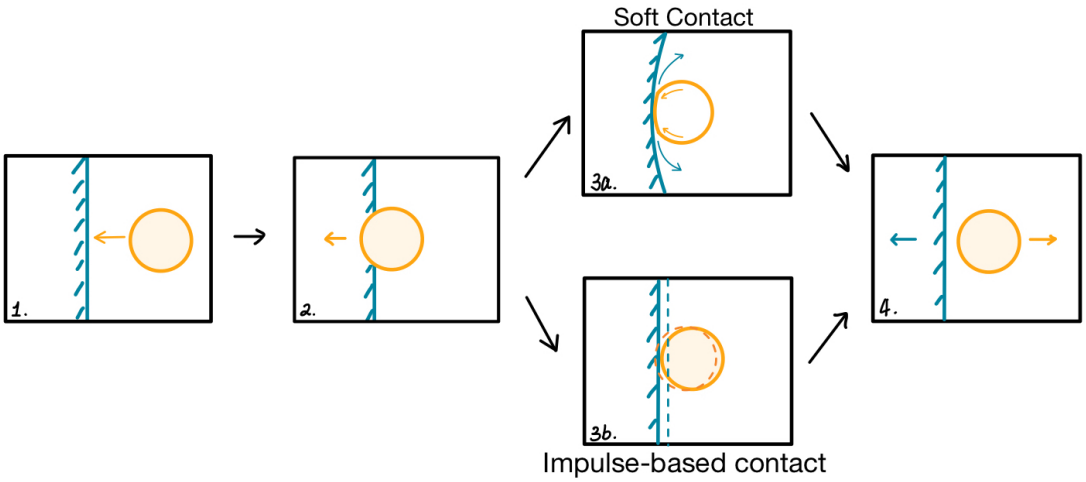


Figure 3.2: Contact resolving in two structurally different dynamics models. From identical initial conditions (1), the pusher contacts the box (2). Under a soft contact model (3a), slight interpenetration is permitted, and contact forces are resolved through constrained optimisation. Under an impulse-based contact model (3b), a strict no-penetration constraint is enforced, and contact is resolved through instantaneous impulses.

3.3 Policy Training

We selected SKRL [25], an open-source RL library, as a unified training backend across both simulators. Using the same implementation in training removes the learning algorithm as a source of variation, so any differences in policy performance can be linked to the simulators and their dynamics models rather than to the training code.

3.3.1 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [24] is a model-free, on-policy reinforcement learning algorithm from the policy gradient family, typically implemented in an actor-critic configuration. It was introduced as a simpler alternative to Trust Region Policy Optimization [23], replacing its hard constraint on policy updates with a clipped surrogate objective. By clipping the ratio between the new and old policy probabilities, PPO prevents any single update from moving the policy too far from its previous behaviour. This improvement allows multiple gradient steps to be taken per batch of collected interactions without the instability that typically affects traditional policy gradient methods.

PPO is well-suited to the box-pushing task studied in this work for several reasons. It handles continuous action spaces natively through a Gaussian policy, which matches the two-dimensional velocity commands used to control the pusher. Its model-free formulation avoids the need to explicitly model the contact-rich dynamics of pushing. As an on-policy algorithm, PPO also works well with Domain Randomisation because the policy is trained on data generated entirely by the current randomised dynamics.

All policies in this work were trained with PPO using a recurrent neural network (RNN) architecture, with configuration informed by prior work within the research group [3]. The training configuration is shown in Table 3.1. Both the policy and value networks share the same architecture, consisting of a fully connected Exponential Linear Unit (ELU) layer, a Long Short-Term Memory (LSTM) layer, and a second fully connected ELU layer. All training runs used 1024 parallel environments.

3.3.2 Training pipelines

To investigate whether exposure to structurally different dynamics models during training improves policy transferability, five training pipelines were designed:

- **Pipeline I:** Single simulation training in Isaac Sim.
- **Pipeline M:** Single simulation training in MuJoCo.
- **Pipeline MI:** Multi-simulation training, start in MuJoCo, then in Isaac Sim.
- **Pipeline IM:** Multi-simulation training, start in Isaac Sim, then in MuJoCo.
- **Pipeline S:** Multi-simulation training, training in both simulations simultaneously.

Pipelines I and M serve as baselines for single-simulator transfer, while MI, IM, and S test whether sequential or simultaneous exposure to multiple structurally different dynamics models improves policy performance.

3.3.3 Parameter randomisation

Since MuJoCo and Isaac Sim use different parameterisation in their contact models; not all physical parameters have a direct mapping between the two. To account for this mismatch, and to investigate how randomisation interacts with structurally different dynamics models, Domain Randomisation was applied to object properties at three

Parameter	Value
Algorithm	PPO
Discount factor (γ)	0.99
GAE lambda (λ)	0.95
Learning rate	3×10^{-4}
Learning rate scheduler	KL Adaptive
KL threshold	0.008
Rollout length	100
Learning epochs	8
Mini-batches	8
Clip ratio	0.2
Value clip	0.2
Gradient norm clip	1.0
Value loss scale	2.0
Entropy loss scale	0.0
Rewards shaper scale	0.5
Policy network	256 (ELU) $\rightarrow$ LSTM(256) $\rightarrow$ 128 (ELU)
Value network	256 (ELU) $\rightarrow$ LSTM(256) $\rightarrow$ 128 (ELU)
Policy output activation	Tanh
Separate policy and value networks	Yes
Parallel environments	1024
Seed	42

Table 3.1: PPO training configuration.

levels. The parameters exposed by each simulator, along with their default values, are discussed in Chapter 4.

- **No Randomisation (NR):** All parameters were kept at their default values, serving as baselines for parameter randomisation.
- **Partial Randomisation (PR):** Randomises the most physically intuitive and directly comparable parameters across simulators – box mass and sliding friction. In Isaac Sim, dynamic and static friction were constrained to the same value per object, effectively reducing the friction model to a single sliding friction coefficient comparable to MuJoCo.
- **Full Randomisation (FR):** Extends into simulator-specific parameters that do not have direct cross-simulator equivalents. On top of the randomised properties in **PR**, in Isaac Sim, dynamic and static friction were allowed to differ per object, and restitution was additionally randomised. In MuJoCo, torsional friction and contact solver parameters (solref) were additionally randomised.

3.4 Performance evaluation

To compare the performance of trained policies, we designed two evaluation domains: the box mass domain and the table-friction domain. Each domain uniformly varies a

single physical property while holding all other parameters at their default values.

The box mass domain uses 250 values, uniformly distributed from 0.01 kg to 2.5 kg. The table-friction domain uses 120 friction coefficients, uniformly distributed from 0.01 to 1.20. Together, these create 740 different dynamic settings across both simulators. A shared set of 100 randomly selected goal positions for the box was used for all settings. Every policy was evaluated on every combination of dynamics setting and goal. Performance was measured by the success rate across the 100 goals.

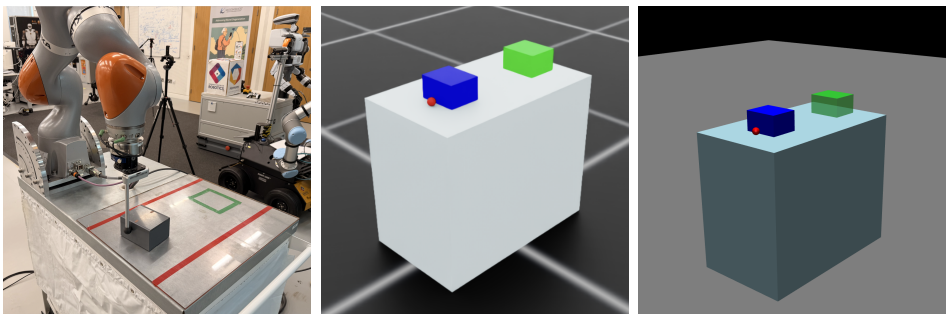
To assess how well policies generalise, we defined two performance regimes. **In-domain performance** refers to evaluation environments whose swept parameter falls within the randomisation range used during training (0.1 kg to 0.6 kg for box mass and 0.2 to 0.8 for table friction coefficient). This is the regime we would expect the policy to handle reliably, as it matches the conditions seen during training. **Out-of-domain performance** refers to all evaluation environments outside these ranges, testing how well the policy generalises beyond the conditions it was trained on.

Chapter 4

Implementation

This chapter describes the engineering implementation of the experimental design outlined in Chapter 3, covering the construction of simulations, the training pipelines, and the evaluation pipelines.

4.1 Simulation Construction

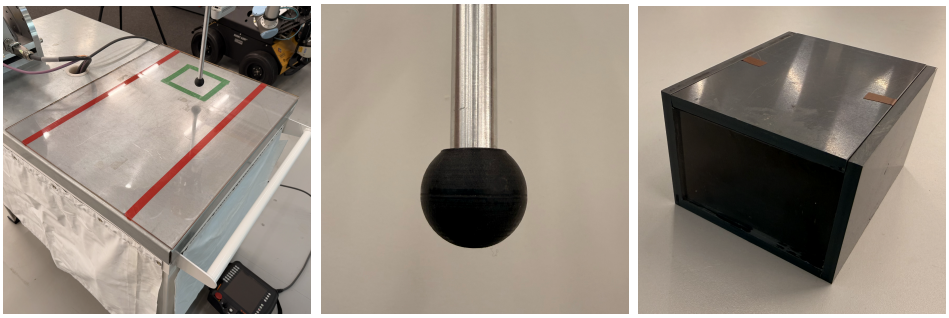


(a) Real World (b) Isaac Sim (c) MuJoCo

Figure 4.1: Task setups

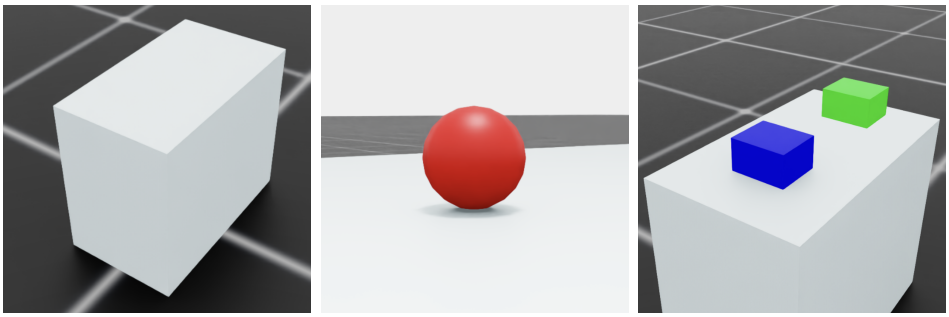
4.1.1 Object Geometries

In both Isaac Sim and MuJoCo, the geometries of the table, box, and pusher were modelled using primitive shapes. Cubes represent the table and the box, while a sphere represents the pusher. Modelling the pusher with only its end effector, rather than the entire robotic arm, was intended to minimise differences in actuation models and isolate the dynamic gap between two simulators.



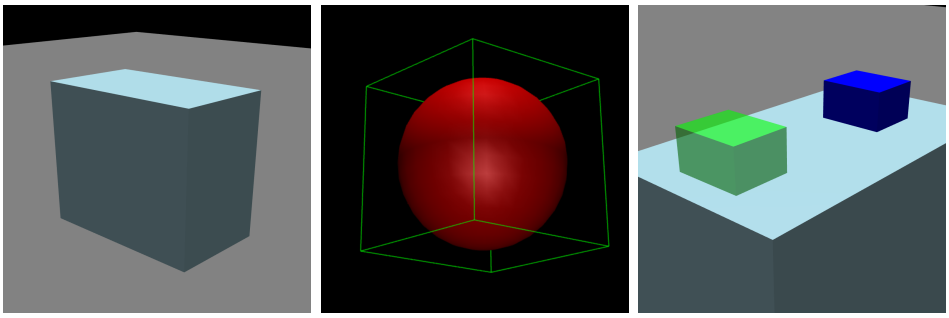
(a) Table (b) Pusher (c) Box

Figure 4.2: Real world objects



(a) Table (b) Pusher (c) Box

Figure 4.3: Object Geometries in Isaac Sim



(a) Table (b) Pusher (c) Box & Goal marker

Figure 4.4: Object Geometries in MuJoCo

4.1.2 Object Parameters

Although objects share the same visual geometry across both simulators, their physical properties are specified differently. This is due to the distinct parameter space in contact models that each simulator uses.

The default parameters selected in Isaac Sim and MuJoCo are shown in Table 4.1 and Table 4.2 respectively. The mass of each object were kept consistent across both simulators. Dynamic and static friction in Isaac Sim and MuJoCo's sliding friction

were set to 0.5, which was a reasonable approximation for plastic surfaces. The default values selected for restitution in Isaac Sim and solref in MuJoCo both describe critically damped contact behaviour between objects.

When objects are in contact, both simulators use the friction and contact parameters of the higher-priority object. In our setup, there are two contact pairs: box-pusher and box-table. We assigned the box a lower priority than both the pusher and the table, so only its mass parameter was used in the simulation. To reduce the influence of differing parameter spaces on policy performance, we introduced randomisation ranges for some parameters.

4.2 Training

In order to enable interactions between the learning agent and the simulated environments, an additional layer was implemented in both simulations to expose the following operations with a unified interface:

- **Reset:** Initialise or reinitialise the environment to a starting configuration. This includes setting the box to the initial pose, placing the pusher at its starting position, and sampling a random target goal pose. Levels of parameter randomisation (subsection 3.3.3) was implemented within this operation by sampling new values for the randomised properties at each reset.
- **Step:** Advance the simulation by one action timestep given an action from the policy. This involves applying the commanded velocity to the pusher, stepping the physics engine, returning the resulting observation, computing the reward and checking for terminal conditions.

Together, these two operations allow the RL agent to interact with the simulation: it receives observations and rewards in response to its actions, and the environment resets when a terminal state is reached. This follows the standard Gymnasium environment interface [29], so the training library can interact with both simulators through the same API.

4.2.1 Training with Isaac Sim

The abstraction layer described above was implemented in Isaac Sim using Isaac Lab [19], the official machine learning framework developed by NVIDIA for Isaac Sim. The complete structure of this pipeline is shown in Figure 4.5.

Isaac Lab utilises the parallel computing capability of a Graphics Processing Unit (GPU) to provide vectorised environments natively, as demonstrated in Figure 4.6. This allowed the agent to interact with all environments through a single step or reset call. The number of parallel environments in Isaac Sim was set to 1024 to accommodate limitations in MuJoCo's parallelisation, discussed in subsection 4.2.2.

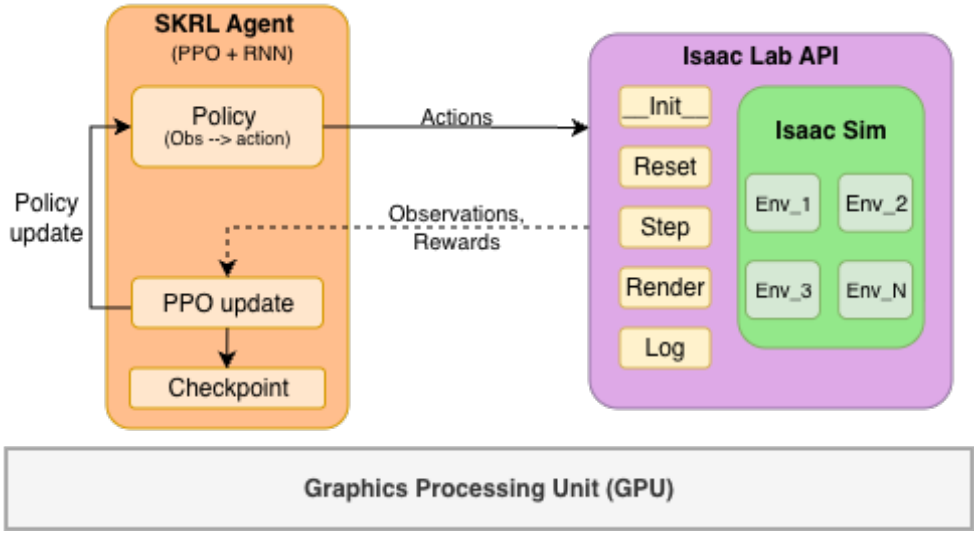


Figure 4.5: Isaac Lab Training pipeline (**I**).

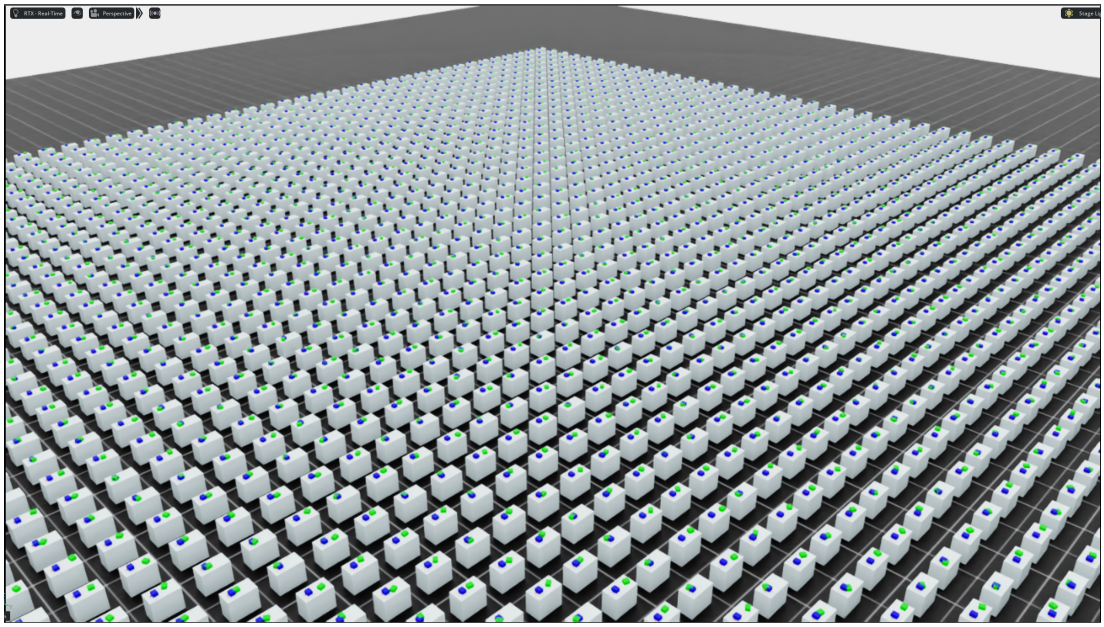


Figure 4.6: Vectorised Environment in Isaac Lab (num_envs = 4096)

4.2.2 Training with MuJoCo

In MuJoCo, the abstraction layer was implemented for a single environment using the Gymnasium API, where each step or reset call acts on one environment only. This is because MuJoCo was designed to perform computations on a Central Processing Unit (CPU), it does not natively support vectorised environments.

To achieve parallelism as demonstrated in Figure 4.8, an additional wrapping layer was implemented with the Gymnasium vectorised environments. The wrapper spawns each environment as a separate CPU subprocess. When a step or reset call is made, the main

process issues asynchronous calls to all subprocesses and waits for their returns before proceeding.

Due to limitations on CPU clock speed and based on experimentation with different configurations, 1024 was selected as the optimal number of parallel environments, balancing both simulation performance and training efficiency. The complete pipeline structure is shown in Figure 4.7.

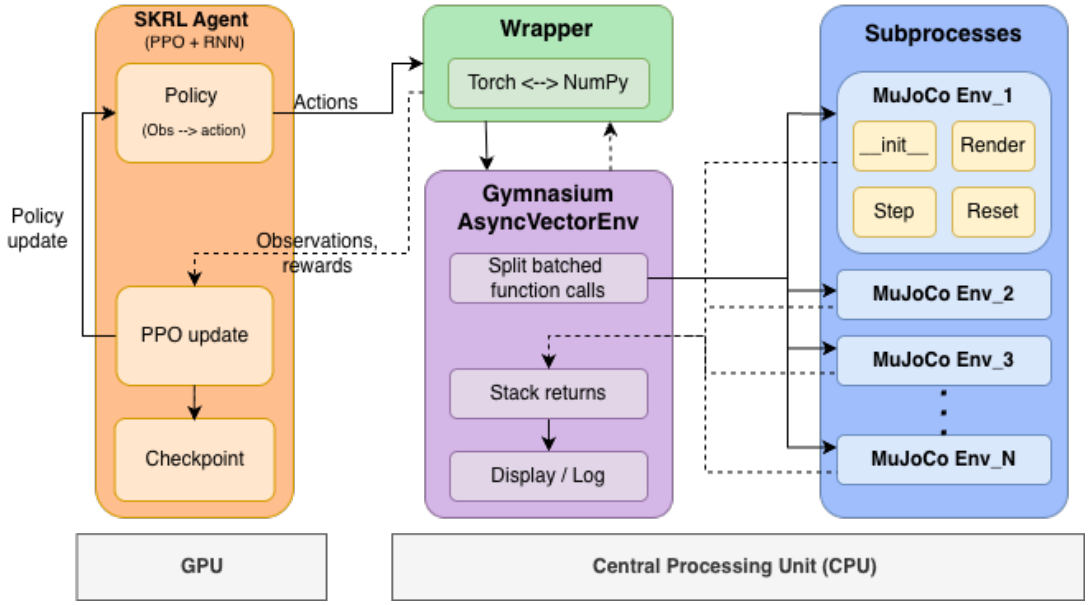


Figure 4.7: Gymnasium + MuJoCo Training pipeline (**M**).

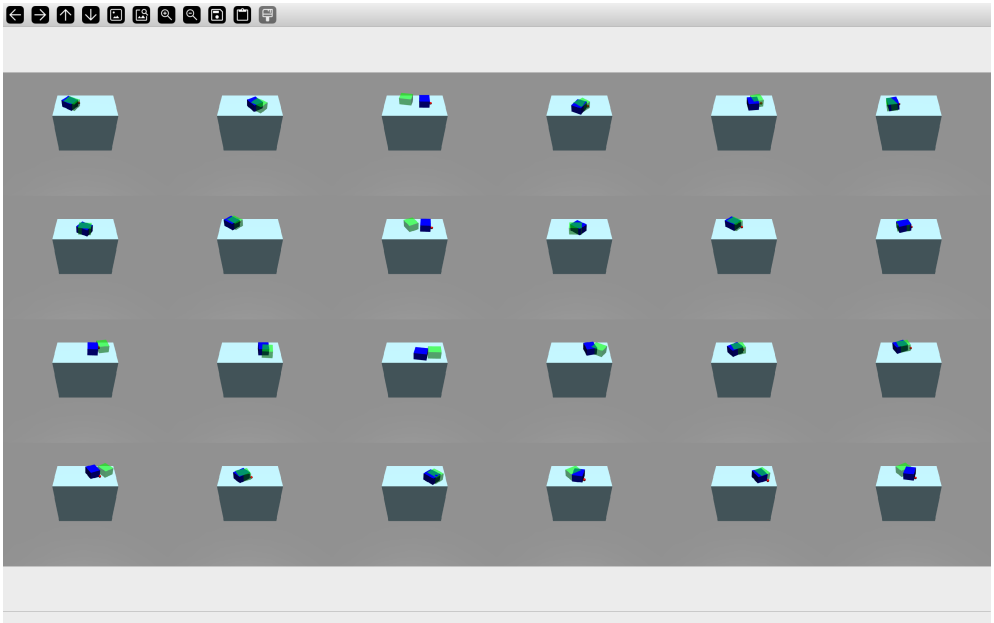


Figure 4.8: Vectorised Environment in MuJoCo (num_envs = 32)

4.2.3 Training across simulators

Since training checkpoints and continual training were supported by the RL library SKRL [25]. Sequential multi-simulator training pipelines (**MI**) and (**IM**) were therefore implemented by saving a checkpoint after training in one simulator and resuming training in the other.

In contrast, the simultaneous training pipeline (**S**), shown in Figure 4.9, requires an additional scheduling layer that initialises the evenly split vectorised environments in both Isaac Sim and MuJoCo within the same process. This layer issues step or reset calls to both environments, waits for their returns, handles data type conversions between GPU tensors and CPU NumPy arrays, and combines the returned values.

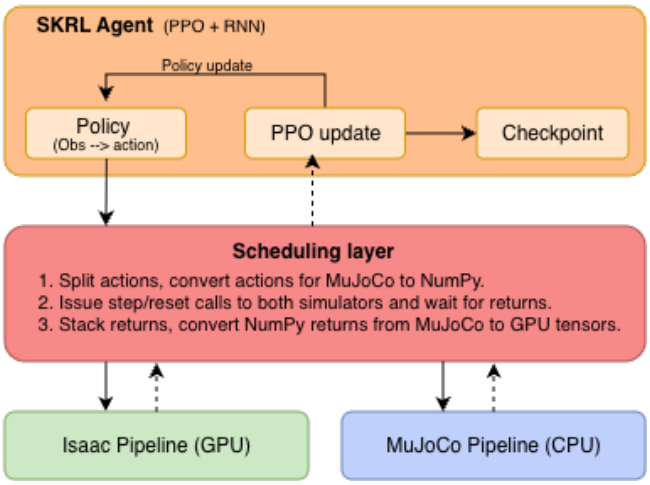


Figure 4.9: Simultaneous Training pipeline (**S**).

4.3 Evaluation pipeline

During evaluation, the SKRL agent loads the final checkpoint produced by one of the training strategies, receives observations, and selects actions through the same interface used during training. The agent running mode was set to “evaluation”, meaning no policy updates were made during this process.

In order to collect performance statistics, three additional components were implemented:

Parameter assigner: This component assigns a uniformly distributed box mass or table sliding friction value to each sub-environment at reset time.

Goal tracker: This component tracks the number of finished evaluations in each sub-environment, and points to the next goal within the evaluation set.

Episode result collector: This component records the terminal state of each sub-environment from the finished evaluation when the *reset* function is called. It was later discovered that this component could also be used to monitor policy performance during trainings, when integrated with Weight & Bias API, as shown in Figure 4.10.



Figure 4.10: Weight & Bias Integration

Object	Property	Default Value	Randomisation Range
Table	Mass	30	N/A
	Dynamic friction	0.5	[0.2, 0.8]
	Static friction	0.5	[0.2, 0.8]
	Restitution	0.0	[0.0, 0.2]
Pusher	Mass	0.5	N/A
	Dynamic friction	0.5	[0.2, 0.8]
	Static friction	0.5	[0.2, 0.8]
	Restitution	0.0	[0.0, 0.2]
Box	Mass	0.5	[0.1, 0.6]
	Dynamic friction	0.0	N/A
	Static friction	0.0	N/A
	Restitution	0.0	N/A

Table 4.1: Isaac object properties

Object	Property	Default Value	Randomisation Range
Table	Sliding friction	0.5	[0.2, 0.8]
	Torsional friction	0.005	N/A
	Rolling friction	0.0001	N/A
	Solref (timeconst)	0.02	[0.005, 0.02]
	Solref (dampratio)	1.0	[0.8, 1.2]
Pusher	Mass	0.5	N/A
	Sliding friction	0.5	[0.2, 0.8]
	Torsional friction	0.005	N/A
	Rolling friction	0.0	N/A
	Solref (timeconst)	0.02	[0.005, 0.02]
	Solref (dampratio)	1.0	[0.8, 1.2]
Box	Mass	0.5	[0.1, 0.6]
	Sliding friction	0.0	N/A
	Torsional friction	0.0	N/A
	Rolling friction	0.0	N/A
	Solref (timeconst)	0.02	N/A
	Solref (dampratio)	1.0	N/A

Table 4.2: MuJoCo object properties

Chapter 5

Experiment Results & Discussion

This chapter presents and analyses the results of the experiments outlined in the preceding chapters. It covers the convergence behaviour of the trained policies, their transferability across simulators under different training configurations, and a finding related to friction parameter mapping that reinforces the distinction between parametric and structural differences in dynamics models.

5.1 Trained policies

Policy Code	Training Timesteps	Agent Code	Training Timesteps
I_NR	300k	M_NR	300k
I_PR	500k	M_PR	500k
I_FR	500k	M_FR	500k
IM_NR	200k+100k	MI_NR	200k+100k
IM_PR	300k+200k	MI_PR	300k+200k
IM_FR	300k+200k	MI_FR	300k+200k
S_NR	500k	S_PR	N/A

Table 5.1: Policies trained

Table 5.1 summarises all policies produced in this work. Each policy is identified by a code indicating the training pipeline and the parameter randomisation level used. All policies trained in a single simulator converged by the end of training, as shown by the stable mean total reward and task success rate in Figure 5.1. Training in pipeline (**M**) for the same number of steps took noticeably longer than in pipeline (**I**) as shown in Figure 5.2. Although this does not directly influence policy performance, it demonstrates GPU's higher parallelism advantage over the CPU.

When training policies without any parameter randomisation, convergence was observed for (**M_NR**) and (**I_NR**) at step 200k. At this stage, checkpoints from both training runs were exchanged: the (**M_NR**) checkpoint was loaded into training pipeline I, and the (**I_NR**) checkpoint was loaded into pipeline M. This transfer enabled the policies to continue training in the new pipelines. Both transferred policies reached convergence

again after 100k steps, resulting in (**MI_NR**) and (**IM_NR**). For comparison, the original policies (**M_NR** and **I_NR**) remained in their respective pipelines and were trained for the same number of steps. The same transfer and continued training procedure was applied to policy training involving a larger number of randomised parameters.

Training in pipeline (**S**) was a special case added later in the experiment. Policy (**S_NR**) failed to converge within 300k timesteps and was therefore trained for 500k timesteps. An attempt was also made to train the policy (**S_PR**); however, it failed to converge under the standard learning configuration of 1024 parallel environments, even with increased training timesteps. While convergence was achieved with 2048 parallelised environments, this result was excluded to maintain consistency across all training configurations.

We hypothesise that the different convergence behaviour of pipeline (**S**) compared to other pipelines was caused by the broader dynamic distribution the policy must fit. Simultaneous training across two structurally different simulators already introduces dynamics-level diversity, and adding parameter randomisation further widens this distribution, thereby increasing the difficulty of convergence.

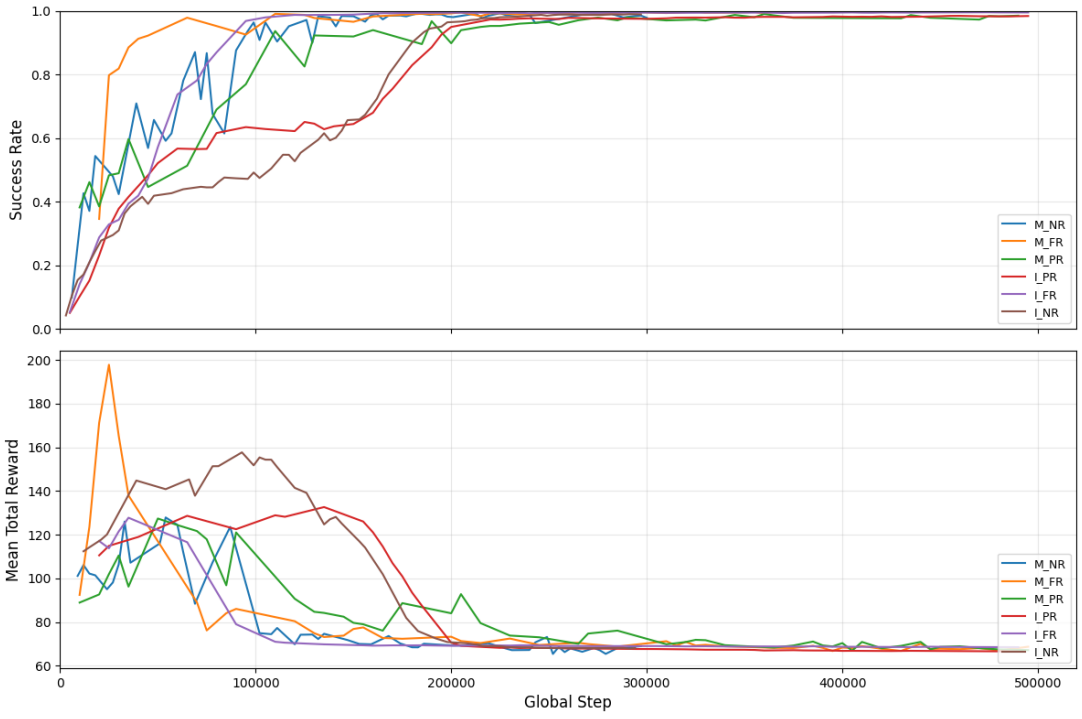


Figure 5.1: Policy total reward (mean) and success rate against number of timesteps during training: all policies achieved convergence at the end of the training.

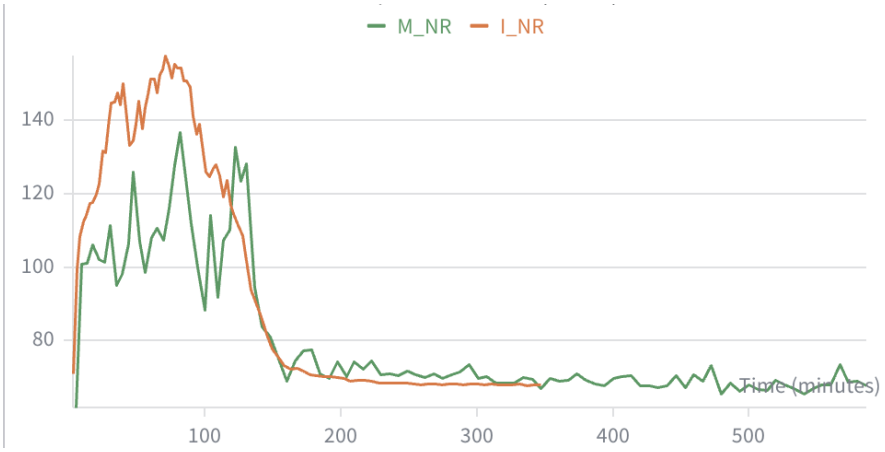
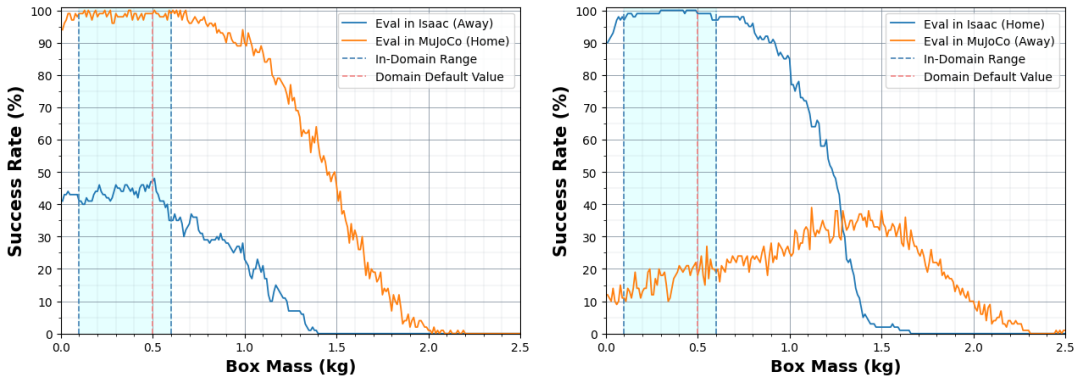


Figure 5.2: Policy total reward (mean) against training runtime for policies **M_NR** and **I_NR**: Training for the same number of steps takes longer in MuJoCo than in Isaac Sim.

5.2 Model Expressiveness



(a) M_NR (b) I_NR

Figure 5.3: Transfer performance of policies M_NR and I_NR.

To investigate policy performance after transfer, we compared success rates in the box mass domain for policies trained in one simulator (the *home* environment) and evaluated in the other (the *away* environment). Figure 5.3 presents this comparison for policies (**M_NR**) and (**I_NR**).

In (**M_NR**)'s evaluation shown in Figure 5.3a, (**M_NR**) in its home MuJoCo environment, it achieves a near 100% success rate within the in-domain range, gradually declining to around 40% at 1.5 kg. When evaluated in the away Isaac Sim environment, performance drops across the entire domain. However, its peak success rate of 48% remains centred within the in-domain range at 0.51 kg, close to the training default of 0.5 kg.

Policy (**I_NR**) shown in Figure 5.3b, exhibits similar home performance in the in-domain range, with a sharp decline to about 40% at approximately 1.24 kg. In the away MuJoCo environment, (**I_NR**) also experiences a significant drop across the domain, but the pattern differs. (**I_NR**)'s peak moved to the out-of-domain region, success rate of 39% shifted to 1.12 kg, away from the training default.

As Table 5.3 shows, (**I_NR**) achieved an above 20% success rate for 8 of the 50 in-domain evaluation points in MuJoCo, compared to all 50 for (**M_NR**) in Isaac Sim. However, across the full domain, (**I_NR**) covers 123 of 250 points, compared to 105 for (**M_NR**). This shows (**I_NR**) has a lower peak performance but a broader coverage after transfer. Shown in Table 5.2 and Table 5.3, it was consistently observed that, in their *away* environments, MuJoCo-trained policies achieve a higher peak but narrower coverage, while Isaac-trained policies achieve a lower peak but broader coverage when given the same level of randomisation.

We hypothesise that this pattern reflects a difference in model expressiveness: the precision with which a contact model's parameter space specifies dynamics. MuJoCo's soft contact model is more expressive, with a richer parameter set that precisely specifies dynamics for a given configuration. As a result, a policy trained in such an environment learns a strategy tightly coupled to a specific dynamic regime. Isaac Sim's impulse-

based contact model uses fewer parameters and is less expressive. This model produces dynamics that are less detailed for a given parameter configuration. Policies trained in Isaac Sim's less-constrained environment may learn a broader but less precise strategy. When transferred to MuJoCo's more expressive dynamics, policies trained in Isaac Sim do not perform best at the box mass on which they were trained. Instead, peak performance shifts to a different mass, where MuJoCo's dynamics happen to match the Isaac Sim dynamics the policy was trained on.

In a practical Sim-to-Real setting, this transfer pattern suggests a trade-off. If the physical properties of the real system are well measured, policies trained in MuJoCo may produce a smaller Sim-to-Real gap due to their stronger performance near the training range. However, if these properties are uncertain or vary across deployments, policies trained in Isaac Sim may achieve better generalisation. This is also supported by observations from the work of Truong et al. [30], which show that policies trained in higher-fidelity simulators can overfit to simulator-specific dynamics, thereby reducing transferability.

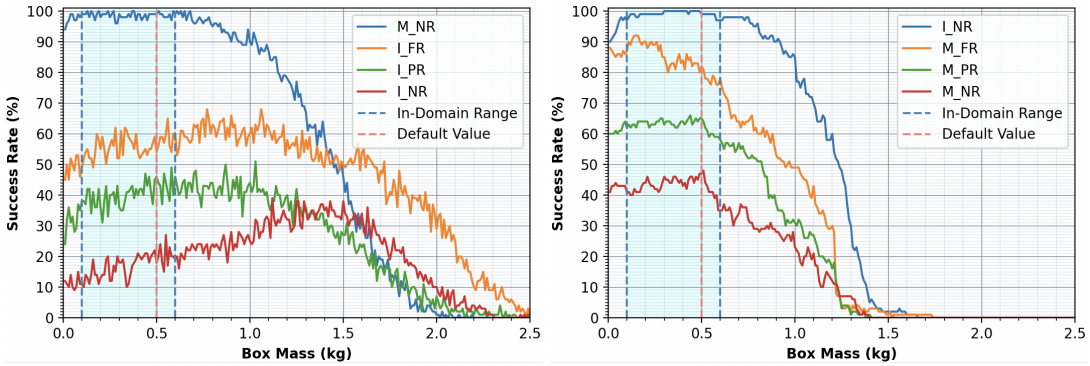
Policy	**Eval Simulator**	**Peak Mass (kg)**	**Peak Rate (%)**	**Mean Rate ± SE (%)**
M_NR	Isaac Sim	0.51	48.0	16.97 ± 1.16
M_PR	Isaac Sim	0.44	66.0	24.68 ± 1.72
M_FR	Isaac Sim	0.14	92.0	33.63 ± 2.29
I_NR (**B**)	Isaac Sim	–	100.0	46.64 ± 2.84
I_NR	MuJoCo	1.12	39.0	19.23 ± 0.69
I_PR	MuJoCo	1.03	51.0	26.49 ± 1.04
I_FR	MuJoCo	0.77	68.0	45.53 ± 1.15
M_NR (**B**)	MuJoCo	–	100.0	56.30 ± 2.63

Table 5.2: Performance of single-simulator policies. Baseline policies are marked with (**B**)

Policy	**Eval Simulator**	**Full Domain > 20%**	**In-Domain > 20%**	**Out-of-Domain > 20%**
M_NR	Isaac Sim	105 / 250	50 / 50	55 / 200
M_PR	Isaac Sim	112 / 250	50 / 50	62 / 200
M_FR	Isaac Sim	121 / 250	50 / 50	71 / 200
I_NR (**B**)	Isaac Sim	133 / 250	50 / 50	83 / 200
I_NR	MuJoCo	123 / 250	8 / 50	115 / 200
I_PR	MuJoCo	164 / 250	50 / 50	114 / 200
I_FR	MuJoCo	210 / 250	50 / 50	160 / 200
M_NR (**B**)	MuJoCo	166 / 250	50 / 50	116 / 200

Table 5.3: Performance distribution of single-simulator policies: number of evaluation points with success rate above 20%. Baseline policies are marked with (**B**)

5.3 Parameter randomisation



(a) Evaluation in MuJoCo (b) Evaluation in Isaac Sim

Figure 5.4: Transferred performance of single-sim policies with different levels of randomisation, compared home policy without any randomisation: Improved transferability by parameter randomisation, but gaps remain.

Figure 5.4 compares the transferred performance of policies trained with a single simulator across three levels of parameter randomisation when compared to native policies trained without any randomisation (I_NR in Isaac, M_NR in MuJoCo). In both graphs, increasing randomisation consistently improves transferred performance. (**FR**) policies outperform (**PR**) policies, which in turn outperform (**NR**) policies. As shown in Table 5.2, the mean transferred success rate improves by 26.3 percentage points from (**I_NR**) to (**I_FR**) in MuJoCo, and by 16.7 percentage points from (**M_NR**) to (**M_FR**) in Isaac Sim.

However, the improvement from increasing randomisation is not symmetric between the two transfer directions. As shown in Table 5.4, the in-domain mean absolute difference (MAD) between the transferred and home success rate curves decreases from 55.96 pp (**M_NR**) to 14.60 pp (**M_FR**) in Isaac Sim. The reverse direction shows less improvement. The in-domain MAD decreases from 81.60 pp (**I_NR**) to 42.70 pp (**I_FR**) when evaluated in MuJoCo. This shows a similar reduction in in-domain MAD, but the in-domain gap between (**I_FR**) and (**M_NR**) is almost three times as wide as (**M_FR**) is from (**I_NR**) when evaluated in Isaac Sim.

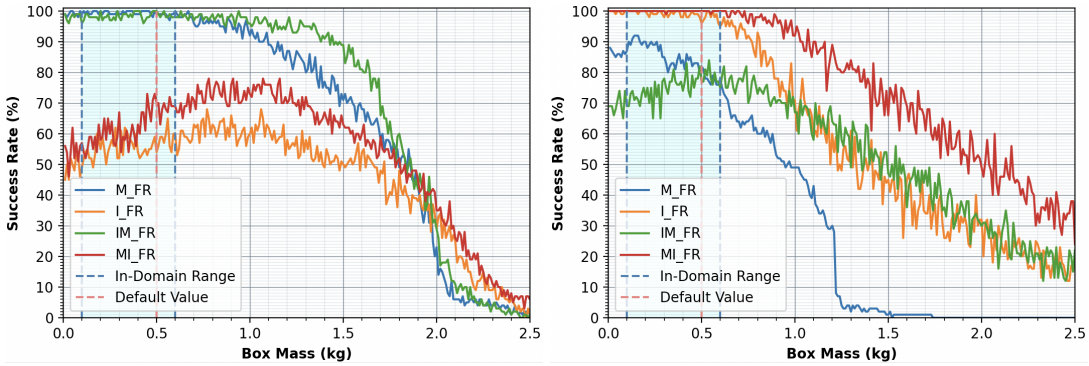
These patterns provide further evidence for our hypothesis in section 5.2. MuJoCo's contact model has a richer parameter space and is more expressive. This allows randomisation to progressively close the transfer gap. In contrast, Isaac Sim's contact model uses fewer parameters and is less expressive. Randomisation in Isaac Sim struggles to precisely generate the contact behaviours that MuJoCo can. This provides evidence that the model's structural differences impose a ceiling on what parameter randomisation alone can achieve.

Chen et al.[6] formally showed that Domain Randomisation can only guarantee transfer when the target environment lies within the simulator's parametric family. When the gap is structural rather than parametric, the probability of sampling the correct dynamics is zero regardless of the randomisation range. This is consistent with our observation.

Policy	Eval Simulator	In-Domain MAD (pp)	Full Domain Mean Deficit (pp)
M_NR	Isaac Sim	55.96	29.67
M_PR	Isaac Sim	36.18	21.96
M_FR	Isaac Sim	14.60	13.04
I_NR	MuJoCo	81.60	39.01
I_PR	MuJoCo	58.38	30.41
I_FR	MuJoCo	42.70	18.88

Table 5.4: Convergence of transferred policies toward home baseline (**I_NR** / **M_NR**). In-domain MAD measures the mean absolute difference within the training parameter range. Mean deficit measures the average shortfall below the home baseline across the full domain, ignoring points where the transferred policy exceeds the baseline.

5.4 Sequential Multi-simulation Training



(a) Evaluation in MuJoCo

(b) Evaluation in Isaac Sim

Figure 5.5: Transferred performance of policies compared against home policy, all with full randomisation: Improved transferability by sequential multi-simulation Training.

Figure 5.5 compares the transferred performance of all policies with full parameter randomisation, against single-simulator baselines also with full randomisation (**FR**).

As shown in Table 5.7, the second simulator clearly shapes policies' in-domain performance. (**IM_FR**) achieved a -0.96 pp in-domain mean difference (MD) when compared against (**M_FR**) in evaluation in MuJoCo. (**MI_FR**) outperforms (**I_FR**) in the in-domain range, achieved +0.88 pp when compared to (**I_FR**) in Isaac Sim. But across the full domain, the sequentially trained policies do not simply copy their single-simulator baseline.

In the MuJoCo evaluation shown in Table 5.5, (**MI_FR**) reaches a mean rate of 53.86% compared to (**I_FR**)'s 45.53%, and (**IM_FR**) improves from (**M_FR**)'s 67.74% to 71.76%. In Figure 5.5a, both sequentially trained policies track the shape of their second simulator's baseline while sitting on top of them, indicating the first training phase provided a consistent lift across the full domain in MuJoCo.

In the Isaac Sim evaluation, (**MI_FR**) achieves a mean success rate of 76.34%, shows

improvement when compared to (**I_FR**)'s 59.66%, and is the only policy to have success rate above 20% in all 250 evaluation points (Table 5.6). (**IM_FR**) improves from (**M_FR**)'s mean success rate of 33.63% to 54.07%, improving out-of-domain coverage from 71 to 180. But the way (**IM_FR**) improves is interesting. Unpacked in Table 5.5, (**IM_FR**) has a lower in-domain MD of -23.78, outperformed by (**M_FR**) when both are compared to (**I_FR**) in Isaac Sim. However, the huge improvement came in out-of-domain performance. Shown in Figure 5.5b, (**IM_FR**) outperformed (**M_FR**) when leaving the in-domain range. When the success rate of (**M_FR**) drops sharply beyond 1.2 kg, (**IM_FR**) maintains substantially higher performance in the rest of the mass range. Rather than simply lifting (**M_FR**)'s profile, (**IM_FR**)'s curve takes on a different shape entirely, suggesting that the broader strategy learned during initial training in Isaac Sim was not fully overwritten by subsequent MuJoCo training.

This behaviour is consistent with findings on sequential training in deep RL, where early training experiences can persist through subsequent training phases despite catastrophic forgetting of the explicit task performance [10]. While the mechanism behind this persistence is beyond the scope of this work, the observation holds: sequential multi-simulator training improves transferability, with the second simulator shaping in-domain behaviour while the first broadens out-of-domain generalisation.

Policy	**Eval Simulator**	**Peak Mass (kg)**	**Peak Rate (%)**	**Mean Rate ± SE (%)**
I_FR	MuJoCo	0.77	68.0	45.53 ± 1.15
MI_FR	MuJoCo	0.78	78.0	53.86 ± 1.32
M_FR (**B**)	MuJoCo	–	100.0	67.74 ± 2.27
IM_FR	MuJoCo	–	100.0	71.76 ± 2.30
S_NR	MuJoCo	–	100.0	78.16 ± 1.78
I_FR (**B**)	Isaac Sim	–	100.0	59.66 ± 1.96
MI_FR	Isaac Sim	–	100.0	76.34 ± 1.47
M_FR	Isaac Sim	0.14	92.0	33.63 ± 2.29
IM_FR	Isaac Sim	0.54	84.0	54.07 ± 1.33
S_NR	Isaac Sim	0.01	99.0	50.16 ± 2.74

Table 5.5: Policy performance: All (**FR**) policies and policy (**S_NR**), baseline policies are marked with (**B**).

Policy	Eval Simulator	Full Domain $> 20\%$	In-Domain $> 20\%$	Out-of-Domain $> 20\%$
I_FR	MuJoCo	210 / 250	50 / 50	160 / 200
MI_FR	MuJoCo	218 / 250	50 / 50	168 / 200
M_FR (**B**)	MuJoCo	199 / 250	50 / 50	149 / 200
IM_FR	MuJoCo	201 / 250	50 / 50	151 / 200
S_NR	MuJoCo	230 / 250	50 / 50	180 / 200
I_FR (**B**)	Isaac Sim	225 / 250	50 / 50	175 / 200
MI_FR	Isaac Sim	250 / 250	50 / 50	200 / 200
M_FR	Isaac Sim	121 / 250	50 / 50	71 / 200
IM_FR	Isaac Sim	230 / 250	50 / 50	180 / 200
S_NR	Isaac Sim	149 / 250	50 / 50	99 / 200

Table 5.6: Number of evaluation points with success rate above 20%: All (**FR**) policies and policy (**S_NR**). Baseline policies are marked with (**B**)

Policy	Eval Simulator	In-Domain MD (pp)	Full Domain Deficit (pp)	Full Domain Improvement (pp)
I_FR	MuJoCo	-43.42	24.00	1.78
MI_FR	MuJoCo	-37.10	16.64	2.76
IM_FR	MuJoCo	-0.96	0.48	4.49
S_NR	MuJoCo	-4.18	1.31	11.73
M_FR	Isaac Sim	-14.48	26.04	0.00
MI_FR	Isaac Sim	+0.88	0.00	16.68
IM_FR	Isaac Sim	-23.78	8.55	2.96
S_NR	Isaac Sim	-1.94	12.08	2.57

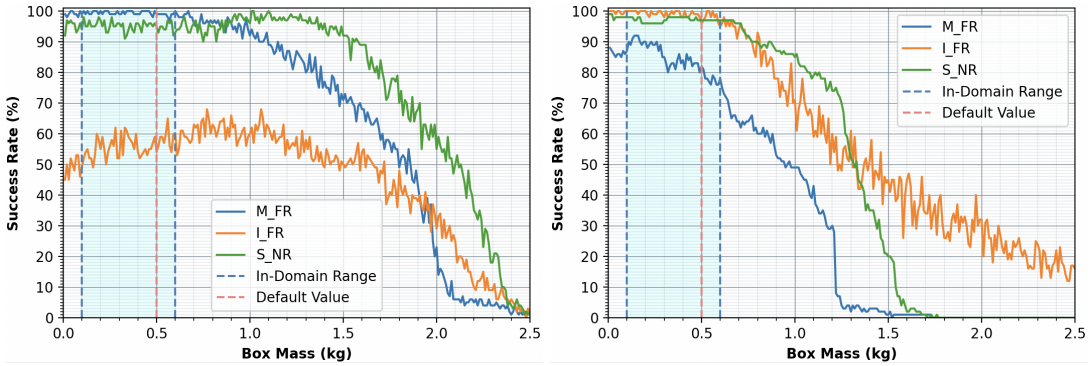
Table 5.7: Policy performance compared against single-simulator home baselines (M_FR in MuJoCo, I_FR in Isaac Sim): In-domain MD is the signed mean difference within the training parameter range. Deficit measures the average shortfall where the policy underperforms the baseline. Improvement measures the average gain where it outperforms.

5.5 Simultaneous Multi-simulation Training

Figure 5.6 compares policy (**S_NR**), trained simultaneously in both simulators with no parameter randomisation, against (**M_FR**) and (**I_FR**), the strongest single-simulator baselines with full randomisation.

In MuJoCo (Figure 5.6a), (**S_NR**) achieved a mean success rate of 78.16%, which exceeds not only (**I_FR**)'s 45.53% but also (**M_FR**)'s home performance of 67.74% (Table 5.5). Data shown in Table 5.7 further confirms this, with (**S_NR**)'s full-domain deficit of just 1.31 pp closely matching the home baseline (**M_FR**), while its improvement of 11.73 pp is the highest of any policy in MuJoCo. (**S_NR**) also has the broadest coverage in MuJoCo with 230 evaluation points above 20% (Table 5.6).

In Isaac Sim (Figure 5.6b), (**S_NR**) achieved a mean rate of 50.16%, above (**M_FR**)'s 33.63% but below (**I_FR**)'s home performance of 59.66%. Its in-domain mean difference of -1.94 pp shows it performs well near the training default, but its out-of-domain



(a) Evaluation in MuJoCo (b) Evaluation in Isaac Sim

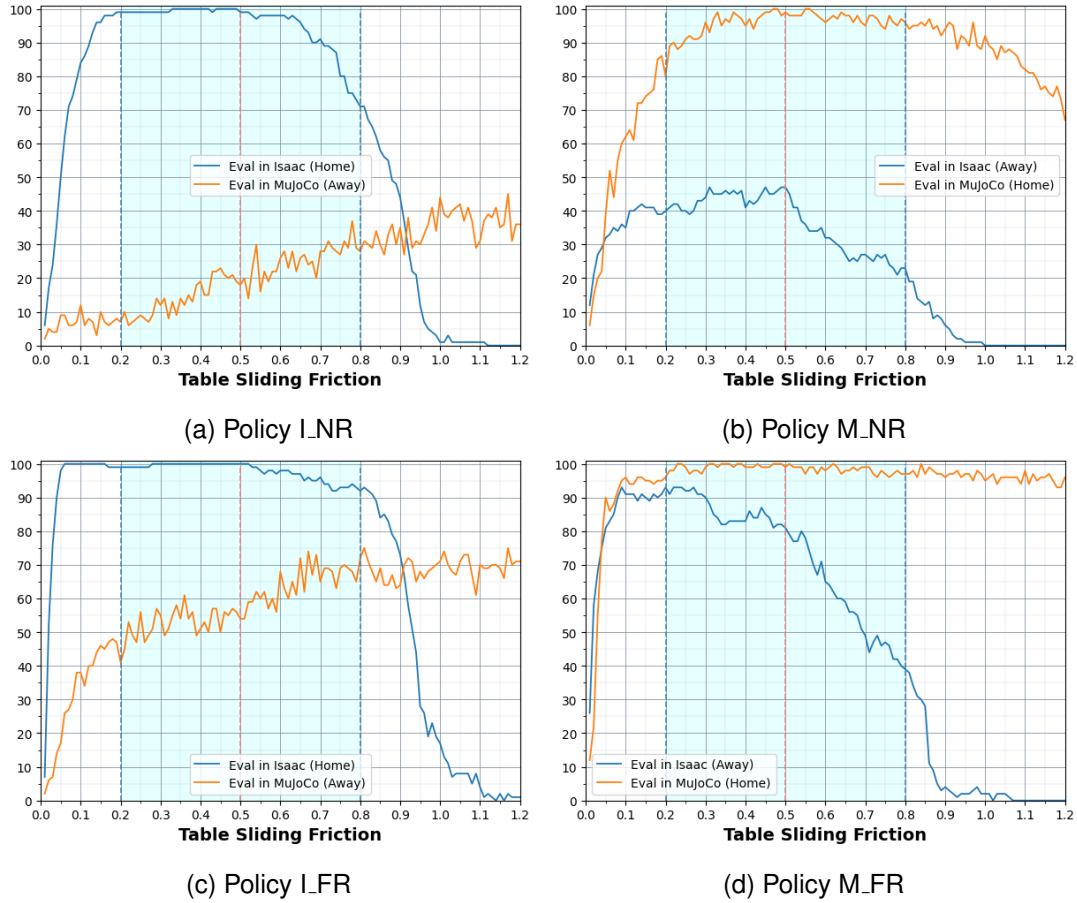
Figure 5.6: Simultaneous Training vs Parameter Randomisation

coverage of 99 is notably lower than (**I_FR**)'s 175 out of 200 evaluation points. Together with full-domain deficit of 12.08 pp, this indicates that (**I_FR**)'s parameter randomisation provides broader gain in out-of-domain performance in Isaac Sim, compared to (**S_NR**)'s simultaneous multi-simulation training.

These results are notable given that (**S_NR**) uses no parameter randomisation at all. Its performance gain came entirely from simultaneous exposure to two structurally different dynamics models. In-domain, this structural diversity produces performance comparable to both baselines trained with full parameter randomisation. This result suggests a complementary relationship between the two approaches. Training in MuJoCo and Isaac Sim at the same time provides structural diversity, enabling (**S_NR**) to learn a strategy that works well in-domain across both simulators; while randomising parameters in a single simulator provides parametric diversity, improving policy performance out of the training domain. Combining both could yield stronger overall performance, though this was not fully evaluated due to the non-convergence of (**S_PR**) under the standard training configuration.

A similar finding was reported by Lei et al. [13], who trained humanoid control policies simultaneously across multiple simulators. Their work showed that this training strategy reduces the influence of a single simulator's modelling assumptions on the final policy. Observations mentioned above support the same conclusion, and further suggest that the structural diversity complements rather than replaces parameter randomisation during policy training.

5.6 Parameter Matching



(a) Policy I_NR

(b) Policy M_NR

(c) Policy I_FR

(d) Policy M_FR

Figure 5.7: Friction Evaluations

Figure 5.7 presents the table friction sweep results for selected policies. A consistent pattern appears across all four plots. In MuJoCo evaluations, both home and transferred policies maintain reasonable performance across the friction range. In Isaac Sim evaluations, however, all policies, including those trained natively in Isaac Sim ((**I_NR**) and (**I_FR**)), exhibit a sharp decline in success rate at higher friction values, eventually failing entirely beyond approximately 0.9.

This pattern reflects a misalignment in how friction is defined across the two simulators. In Isaac Sim, both dynamic and static friction are set to the swept value, meaning the effective friction at each contact involves two separate parameters that increase simultaneously. MuJoCo's single sliding friction coefficient does not produce this effect. The assumption that setting dynamic and static friction to the same value as MuJoCo's sliding friction creates equivalent contact behaviour, while conceptually reasonable, does not hold in practice at higher values.

This finding does not invalidate the evaluation results from the box mass sweep, where the physical quantity has a one-to-one mapping across both simulators. The transferability conclusions drawn in the preceding sections are therefore presented in the box mass domain, and the friction results instead serve as an additional finding, demonstrating

that aligning parameters across structurally different models by conceptual equivalence is insufficient. This reinforces the central argument of this work that the dynamic gap is not only about parameter values, but also about model structure.

Chapter 6

Conclusions

6.1 Summary

This project set out with three aims, each addressed by one of the main findings.

The first aim was to characterise how policies trained in simulators with structurally different contact models behave when transferred across simulators. Our results show that the performance drop after transfer is asymmetric. Policies trained in MuJoCo retain peak performance near the training range when transferred to Isaac Sim; while policies trained in Isaac Sim exhibit lower, shifted peaks but broader coverage when transferred to MuJoCo. We hypothesise that this asymmetry reflects a difference in model expressiveness, where MuJoCo's more expressive contact model produces more precisely defined dynamics for a given set of parameter values, while Isaac Sim's simpler impulse-based model produces less tightly defined dynamics that generalise more broadly but less precisely.

The second aim was to examine whether the effectiveness of Domain Randomisation depends on the expressiveness of the training simulator's contact model. Our results show that varying parameters during training improves transferability in both directions, but is constrained by a structural ceiling imposed by the training simulator's contact model. Randomising within MuJoCo's more expressive model progressively reduced the gap to Isaac Sim, while randomising within Isaac Sim's less expressive model produced similar absolute improvements but did not close the gap to MuJoCo's more precise dynamics. This supports the findings from previous work by Chen et al. [6] that Domain Randomisation can only close the transfer gap when the target dynamics lie within the simulator's parametric family.

The third aim was to explore whether multi-simulator training improves transferability beyond single-simulator training with Domain Randomisation. Our results show that it does. Sequential multi-simulator training produces policies whose in-domain behaviour is determined by the second training phase, while the first phase broadens out-of-domain generalisation. Simultaneous multi-simulator training, even without Domain Randomisation, matches or exceeds single-simulator training with randomisation during transfers. This suggests that the structural diversity offered by training across both

simulators complements the parametric diversity offered by randomisation within a single simulator. The two forms of diversity address different dimensions of the dynamic gap.

A supplementary finding from the friction parameter sweep further reinforces the central argument of this work, showing that aligning parameters across structurally different contact models based on conceptual equivalence is insufficient.

6.2 Limitations

The following limitations of this work should be noted:

- All evaluations were conducted in computer simulations rather than on physical hardware. While the structural distinction between MuJoCo and Isaac Sim provides useful insight into the Sim-to-Real gap, the findings have not been validated in the real world, and conclusions about real-world transferability are indirect.
- Experiments were limited to a single non-prehensile box-pushing task. Whether the observed asymmetric transfer pattern and the benefits of multi-simulator training extend to other manipulation tasks, such as grasping, insertion, or in-hand manipulation, remains an open question.
- Policies in this work were solely trained with Proximal Policy Optimization (PPO). The influence of the training algorithm on policy transfer remains unexplored.
- The benefit of combining simultaneous multi-simulator training with parameter randomisation could not be fully evaluated because policy (**S_PR**) failed to converge under the standard training configuration.
- The random seed used during training does not guarantee that all policies were exposed to the same training samples in the same order. While the large number of training timesteps is expected to reduce the influence of this variation, whether it affects the evaluation results remains unclear.

6.3 Future work

As mentioned previously, one of the possible extensions would be real-world validation. Deploying the trained policies on physical hardware would test whether the asymmetric transfer pattern and the benefits of multi-simulator training observed in this work also hold in the Sim-to-Real setting. A particularly interesting question is whether simultaneous multi-simulator training, which showed strong performance without parameter randomisation, provides a more reliable approach for real-world deployment than parameter randomisation within a single simulator.

Another extension would be repeating the experiments on other manipulation tasks or with different learning algorithms. Box-pushing is a relatively simple contact-rich task with a low-dimensional action space. Tasks such as grasping, insertion,

or in-hand manipulation involve more complex contact behaviour and higher action dimensions, and would test whether the same findings generalise beyond pushing. Similarly, repeating the experiments with different RL algorithms would test whether the observed results are specific to PPO or reflect more general properties of policy learning across structurally different simulators.

A third direction is to resolve the non-convergence of (**S_PR**) by adjusting the learning configuration, such as increasing the number of parallel environments, which was shown to enable convergence, and repeating the experiment. This would provide further evidence for the complementary relationship between simultaneous multi-simulator training and Domain Randomisation proposed in this work.

Bibliography

[1] Jad Abou-Chakra, Lingfeng Sun, Krishan Rana, Brandon May, Karl Schmeckpeper, Niko Suenderhauf, Maria Vittoria Minniti, and Laura Herlant. Real-is-sim: Bridging the sim-to-real gap with a dynamic digital twin, 2025.

[2] Elie Aljalbout, Jiaxu Xing, Angel Romero, Iretiayo Akinola, Caelan Reed Garrett, Eric Heiden, Abhishek Gupta, Tucker Hermans, Yashraj Narang, Dieter Fox, Davide Scaramuzza, and Fabio Ramos. The reality gap in robotics: Challenges, solutions, and best practices, 2025.

[3] Marina Y. Aoyama, Joao Moura, Juan Del Aguila Ferrandis, and Sethu Vijayakumar. Poke and strike: Learning task-informed exploration policies, 2025.

[4] Ian M Bullock, Raymond R Ma, and Aaron M Dollar. A hand-centric classification of human and robot dexterous manipulation. *IEEE transactions on Haptics*, 6(2):129–144, 2012.

[5] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience, 2019.

[6] Xiaoyu Chen, Jiachen Hu, Chi Jin, Lihong Li, and Liwei Wang. Understanding domain randomization for sim-to-real transfer, 2022.

[7] Se-Joon Chung and Nancy Pollard. Predictable behavior during contact simulation: a comparison of selected physics engines. *Computer Animation and Virtual Worlds*, 27(3-4):262–270, 2016.

[8] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021.

[9] Longchao Da, Justin Turnau, Thirulogasankar Pranav Kutralingam, Alvaro Velasquez, Paulo Shakarian, and Hua Wei. A survey of sim-to-real methods in rl: Progress, prospects and challenges with foundation models, 2025.

[10] Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026):768–774, 2024.

[11] Jiabao Dong, Zidi Jia, Yuqing Wang, and Lei Ren. A digital twin framework for

robot grasp and motion generation. In *2024 IEEE Smart World Congress (SWC)*, pages 2045–2052, 2024.

[12] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404, 2015.

[13] Zixing Lei, Zibo Zhou, Sheng Yin, Yueru Chen, Qingyao Xu, Weixin Li, Yunhong Wang, Bowei Tang, Wei Jing, and Siheng Chen. Polysim: Bridging the sim-to-real gap for humanoid control via multi-simulator dynamics randomization, 2025.

[14] Yongkui Liu, He Xu, Ding Liu, and Lihui Wang. A digital twin-based sim-to-real transfer for deep reinforcement learning-enabled industrial robot grasping. *Robotics and Computer-Integrated Manufacturing*, 78:102365, 2022.

[15] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. Active domain randomization, 2019.

[16] Marius Memmel, Andrew Wagenmaker, Chuning Zhu, Patrick Yin, Dieter Fox, and Abhishek Gupta. Asid: Active exploration for system identification in robotic manipulation, 2024.

[17] Fabio Muratore, Christian Eilers, Michael Gienger, and Jan Peters. Data-efficient domain randomization with bayesian optimization. *IEEE Robotics and Automation Letters*, 6(2):911–918, April 2021.

[18] Fabio Muratore, Felix Treede, Michael Gienger, and Jan Peters. Domain randomization for simulation-based policy optimization with transferability assessment. In *Conference on Robot Learning*, pages 700–713. PMLR, 2018.

[19] NVIDIA, :, Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, Milad Rakhsha, Alain Denzler, Eric Heiden, Ales Borovicka, Ossama Ahmed, Iretiayo Akinola, Abrar Anwar, Mark T. Carlson, Ji Yuan Feng, Animesh Garg, Renato Gasoto, Lionel Gulich, Yijie Guo, M. Gussert, Alex Hansen, Mihir Kulkarni, Chenran Li, Wei Liu, Viktor Makoviychuk, Grzegorz Malczyk, Hammad Mazhar, Masoud Moghani, Adithyavairavan Murali, Michael Noseworthy, Alexander Poddubny, Nathan Ratliff, Welf Rehberg, Clemens Schwarke, Ritvik Singh, James Latham Smith, Bingjie Tang, Ruchik Thaker, Matthew Trepte, Karl Van Wyk, Fangzhou Yu, Alex Millane, Vikram Ramasamy, Remo Steiner, Sangeeta Subramanian, Clemens Volk, CY Chen, Neel Jawale, Ashwin Varghese Kuruttukulam, Michael A. Lin, Ajay Mandlekar, Karsten Patzwaldt, John Welsh, Huihua Zhao, Fatima Anes, Jean-Francois Lafleche, Nicolas Moënne-Loccoz, Soowan Park, Rob Stepinski, Dirk Van Gelder, Chris Amevor, Jan Carius, Jumyung Chang, Anka He Chen, Pablo de Heras Ciechomski, Gilles Daviet, Mohammad Mohajerani, Julia von Muralt, Viktor Reutskyy, Michael Sauter, Simon Schirm, Eric L. Shi, Pierre Terdiman, Kenny Vilella, Tobias Widmer, Gordon Yeoman, Tiffany Chen, Sergey Grizan, Cathy Li, Lotus Li, Connor Smith, Rafael Wiltz, Kostas Alexis, Yan Chang, David Chu, Linxi "Jim" Fan,

Farbod Farshidian, Ankur Handa, Spencer Huang, Marco Hutter, Yashraj Narang, Soha Pouya, Shiwei Sheng, Yuke Zhu, Miles Macklin, Adam Moravanszky, Philipp Reist, Yunrong Guo, David Hoeller, and Gavriel State. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning, 2025.

[20] NVIDIA. Physx sdk documentation, Jul 2025.

[21] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2019.

[22] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, page 3803–3810. IEEE, May 2018.

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[25] Antonio Serrano-Muñoz, Dimitris Chrysostomou, Simon Bøgh, and Nestor Arana-Arexolaleiba. skrl: Modular and flexible library for reinforcement learning, 2022.

[26] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[27] Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061, 2014.

[28] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[29] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2025.

[30] Joanne Truong, Max Rudolph, Naoki Yokoyama, Sonia Chernova, Dhruv Batra, and Akshara Rai. Rethinking sim2real: Lower fidelity simulation leads to higher sim2real transfer in navigation, 2022.

Appendix A

Appendix

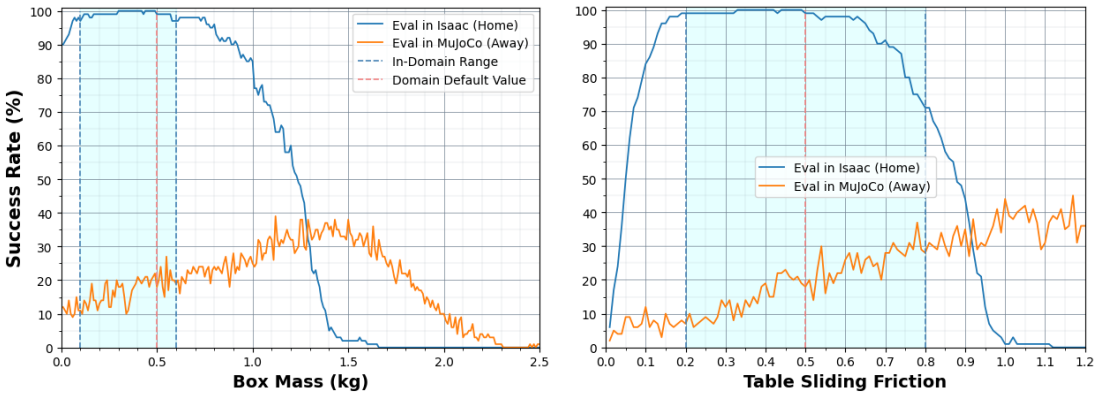
A.1 GitHub Repositories

- Simulation (Isaac Sim)
- Simulation (MuJoCo)
- Trained Policies & Raw Evaluation Result

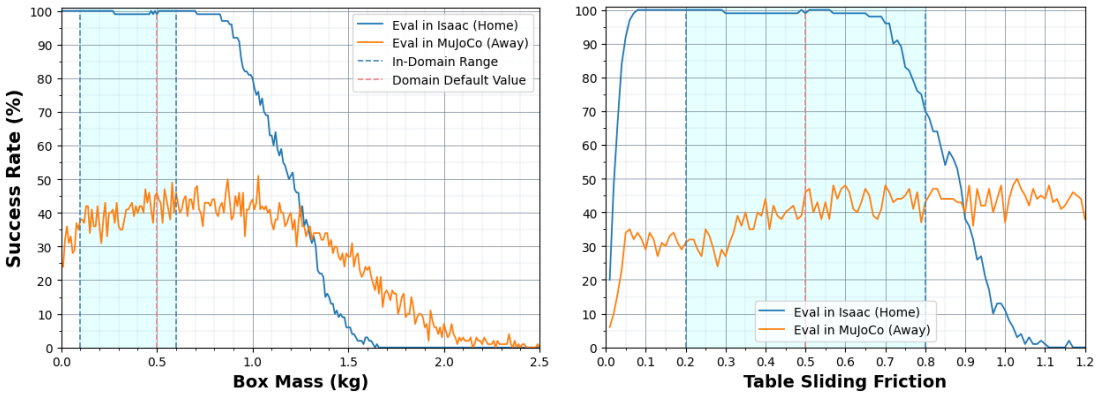
A.2 Policy evaluation results

A.2.1 Home VS Away

(a) I_NR



(b) I_PR



(c) I_FR

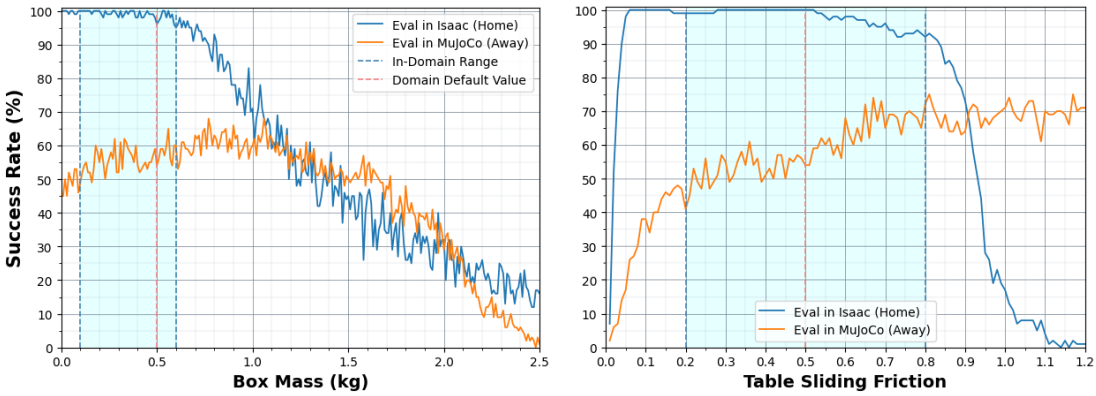
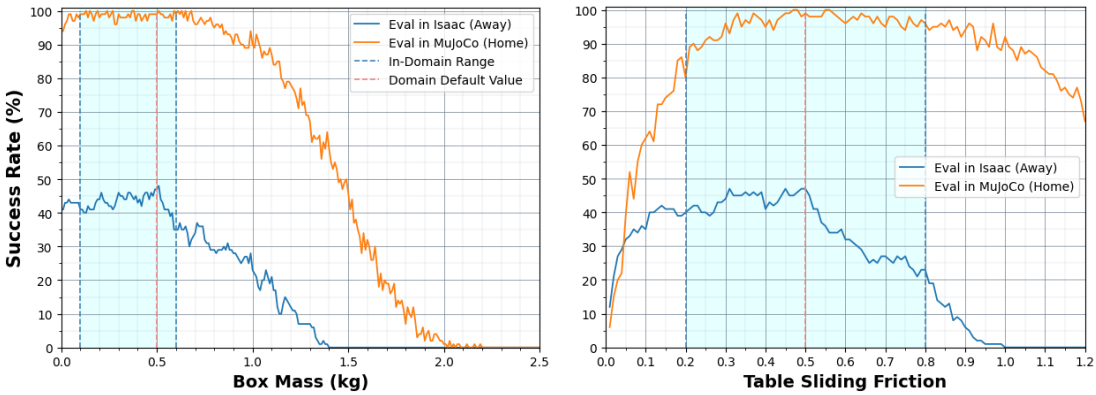
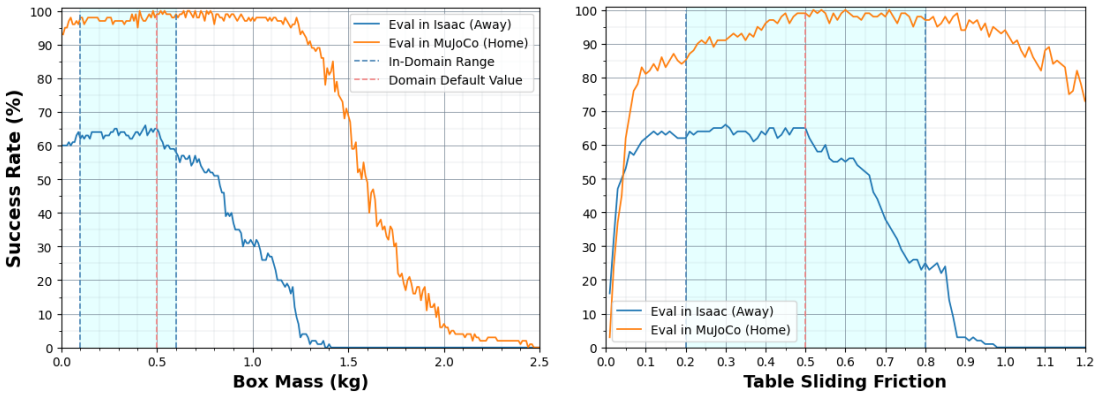


Figure A.1: Benchmark Result (continued on next pages)

(d) M_NR



(e) M_PR



(f) M_FR

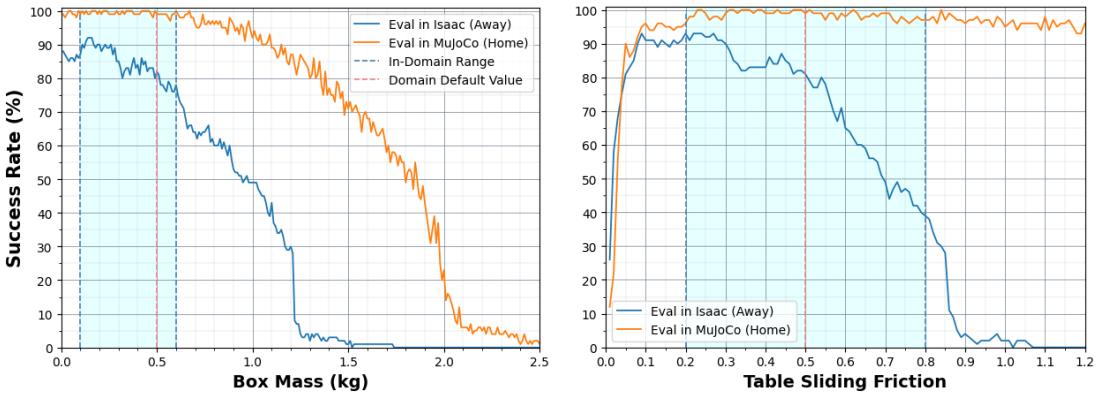
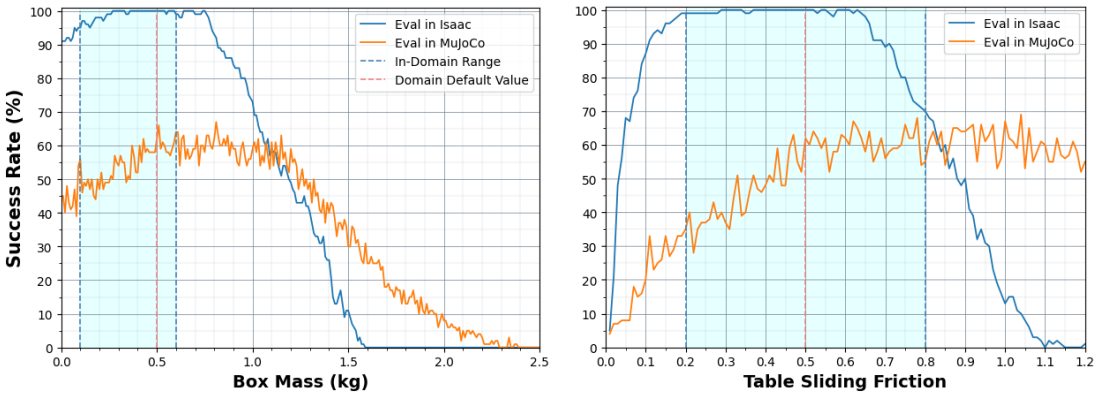
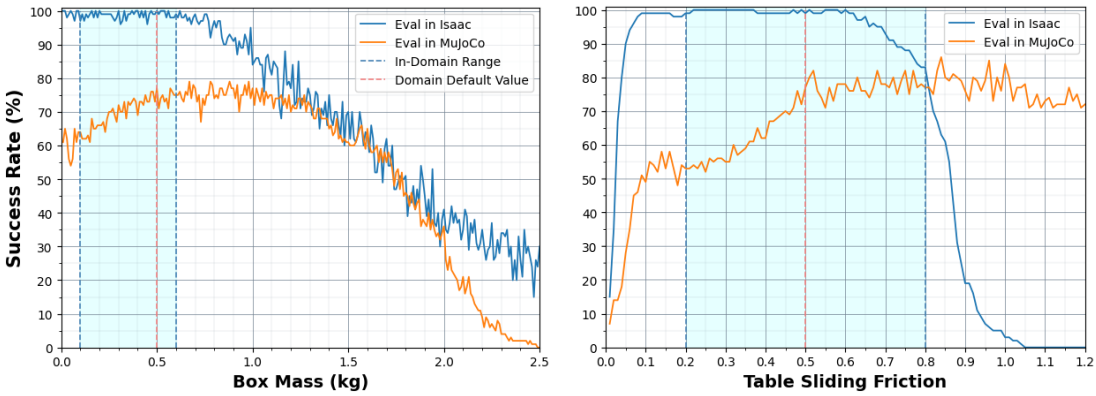


Figure A.1: Benchmark Result (cont.)

(g) MI_NR



(h) MI_PR



(i) MI_FR

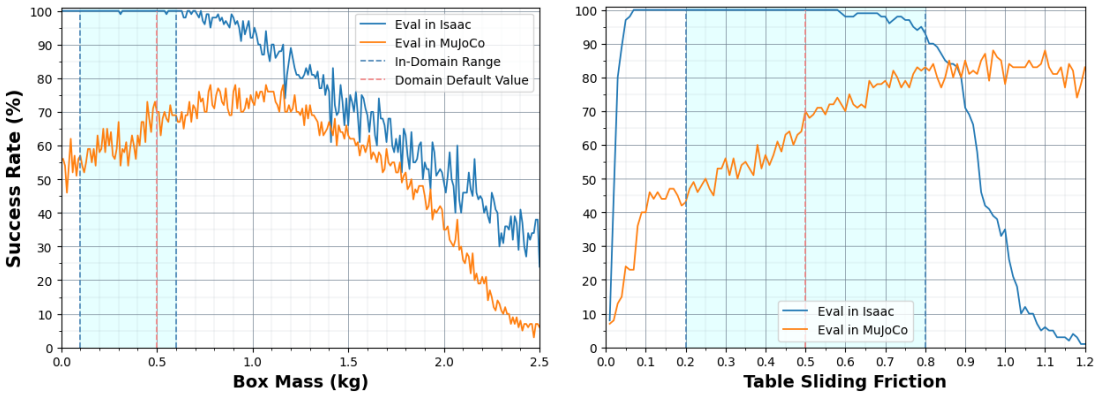
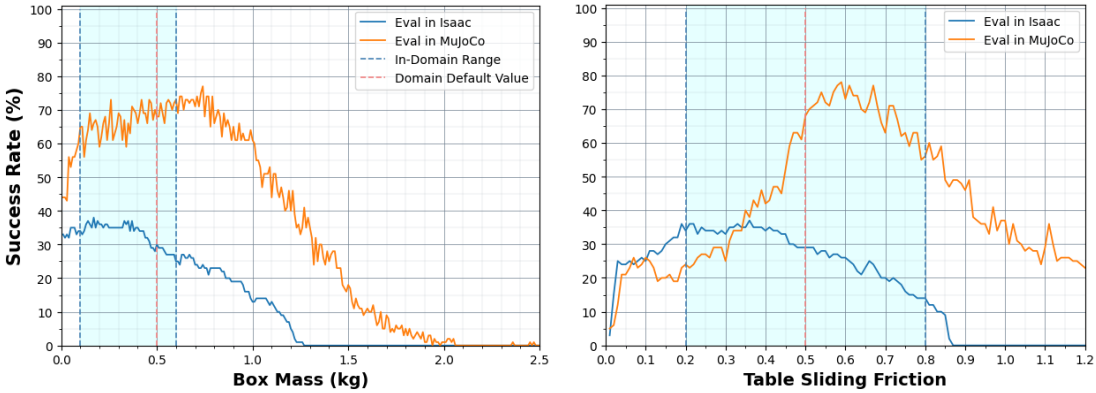
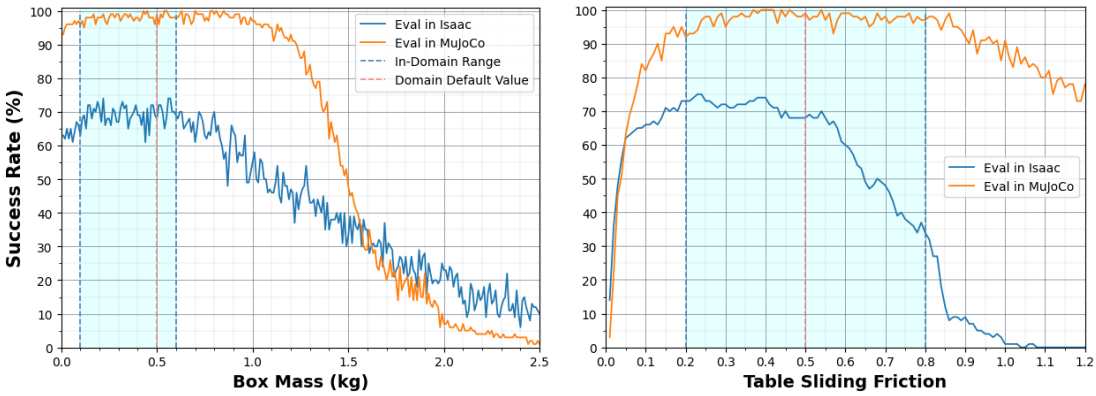


Figure A.1: Benchmark Result (cont.)

(j) IM_NR



(k) IM_PR



(l) IM_FR

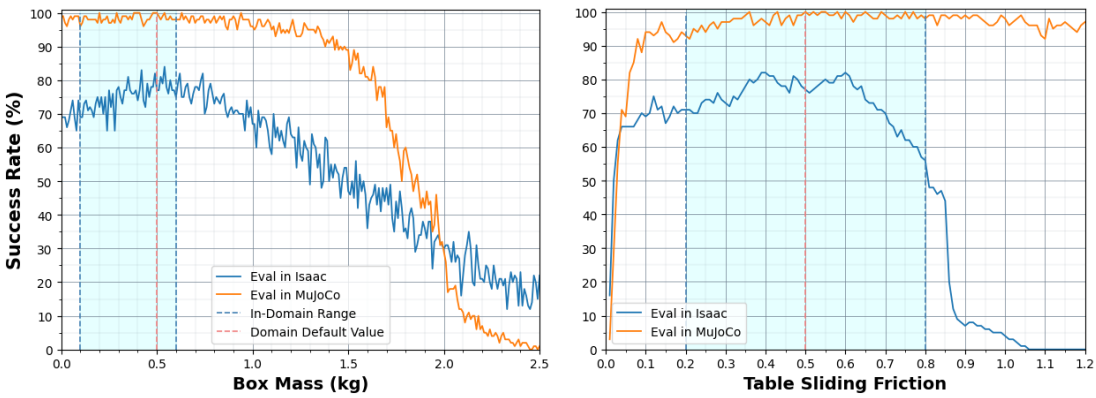


Figure A.1: Benchmark Result (cont.)

(m) S_NR

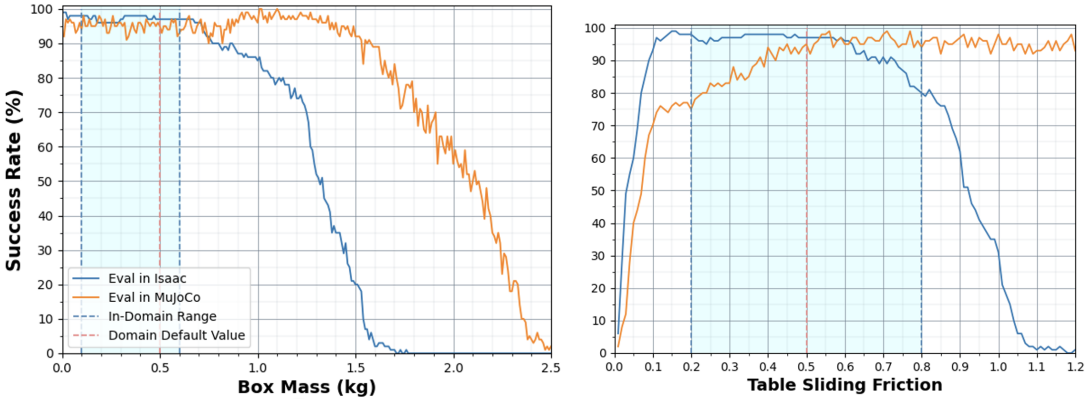
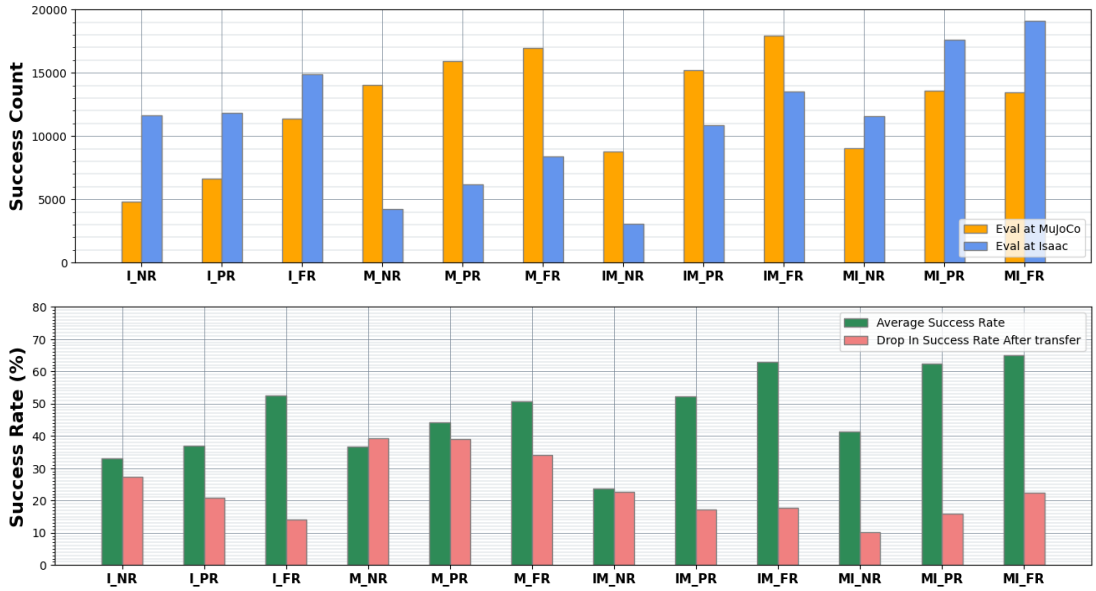


Figure A.1: Benchmark Result (cont.)

A.2.2 By Domain

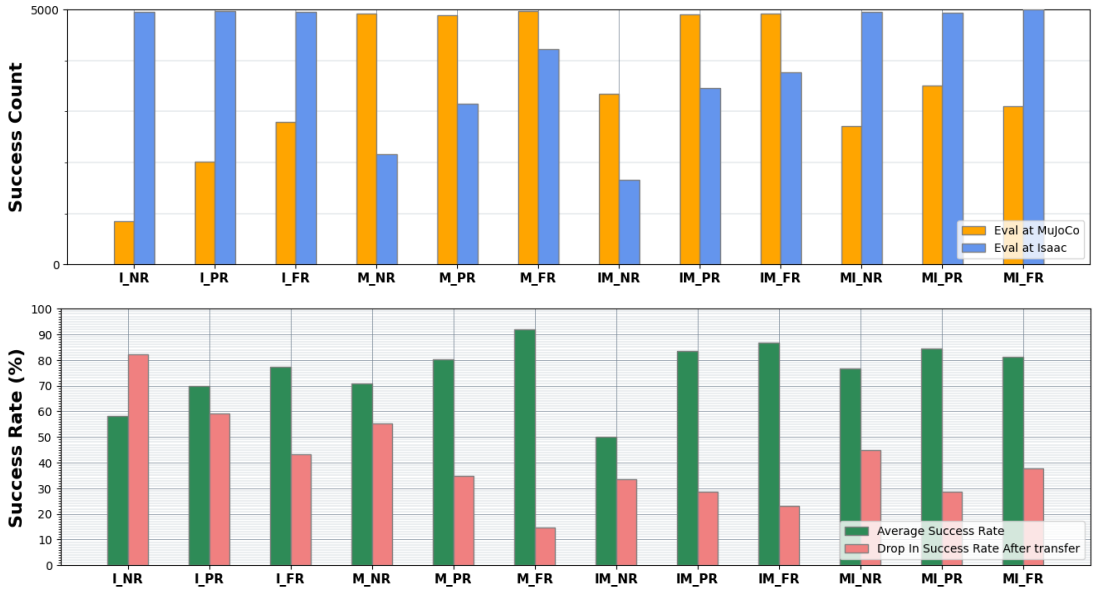
(a) Policy Performance: $BoxMass \in [0.01, 2.50]$



Policy Code	Success Count (MuJoCo) Out Of 250000	Success Count (Isaac) Out Of 250000	Success Rate	Drop After Transfer
I_NR	4807	11660	32.93%	27.41%
I_PR	6623	11832	36.91%	20.84%
I_FR	11382	14916	52.60%	14.14%
M_NR	14076	4242	36.64%	39.34%
M_PR	15911	6170	44.16%	38.96%
M_FR	16936	8407	50.69%	34.12%
IM_NR	8755	3090	23.69%	22.66%
IM_PR	15206	10890	52.19%	17.26%
IM_FR	17939	13517	62.91%	17.69%
MI_NR	9043	11598	41.28%	10.22%
MI_PR	13603	17609	62.42%	16.02%
MI_FR	13465	19086	65.10%	22.48%
Lowest	I_NR (4807)	IM_NR (3090)	IM_NR (23.69%)	MI_NR (10.22%)
Highest	IM_FR (17939)	MI_FR (19086)	MI_FR(65.10%)	M_NR (39.34%)

Figure A.2: Benchmark Result By Domain (Continue next page)

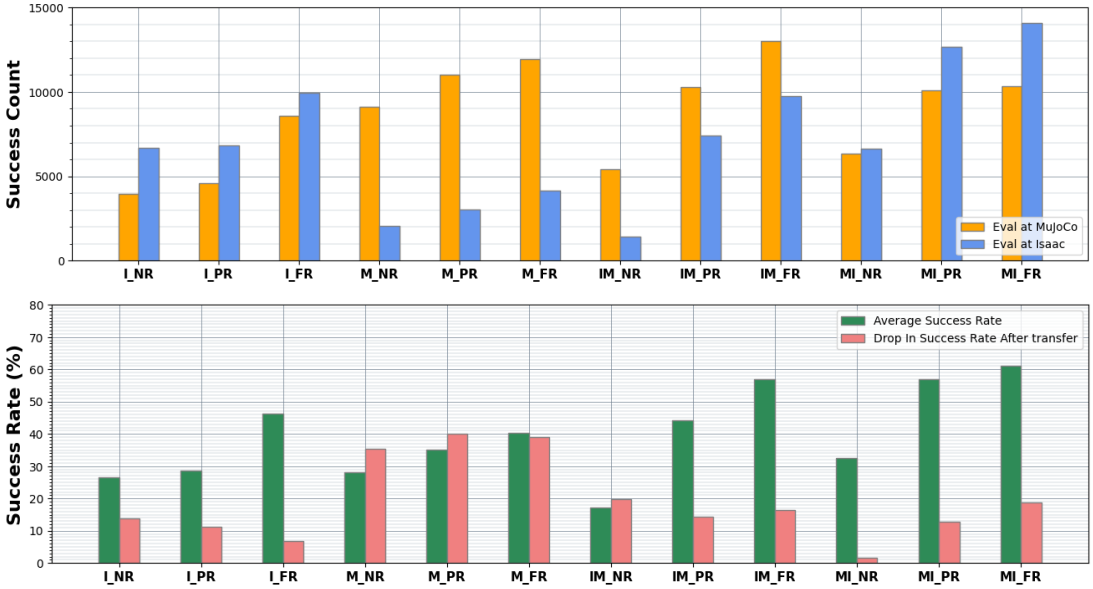
(b) Policy In-Domain Performance: $BoxMass \in [0.10, 0.60]$



Policy Code	Success Count (MuJoCo) Out Of 50000	Success Count (Isaac) Out Of 50000	Success Rate	Drop After Transfer
I_NR	851	4960	58.11%	82.18%
I_PR	2012	4979	69.91%	59.34%
I_FR	2796	4954	77.50%	43.16%
M_NR	4931	2162	70.93%	55.38%
M_PR	4892	3151	80.43%	34.82%
M_FR	4967	4230	91.97%	14.74%
IM_NR	3345	1662	50.07%	33.66%
IM_PR	4903	3462	83.65%	28.82%
IM_FR	4919	3765	86.84%	23.08%
MI_NR	2710	4950	76.60%	44.80%
MI_PR	3510	4939	84.49%	28.58%
MI_FR	3112	4998	81.10%	37.72%
Lowest	I_NR (851)	IM_NR (1662)	IM_NR (50.07%)	M_FR (14.74%)
Highest	M_FR (4967)	MI_FR (4998)	M_FR (91.97%)	I_NR (82.18%)

Figure A.2: Benchmark Result By Domain (cont.)

(c) Policy Out-Of-Domain Performance: $BoxMass \in [0.01, 0.10) \cup (0.60, 2.50]$



Policy Code	Success Count (MuJoCo) Out Of 200000	Success Count (Isaac) Out Of 200000	Success Rate	Drop After Transfer
I_NR	3956	6700	26.64%	13.72%
I_PR	4611	6853	28.66%	11.21%
I_FR	8586	9962	46.37%	6.88%
M_NR	9145	2080	28.06%	35.33%
M_PR	11019	3019	35.09%	40.00%
M_FR	11969	4177	40.37%	38.96%
IM_NR	5410	1428	17.09%	19.91%
IM_PR	10303	7428	44.33%	14.37%
IM_FR	13020	9752	56.93%	16.34%
MI_NR	6333	6648	32.45%	1.57%
MI_PR	10093	12670	56.91%	12.88%
MI_FR	10353	14088	61.10%	18.67%
Lowest	I_NR (3956)	IM_NR (7428)	IM_NR (17.09%)	MI_NR (1.57%)
Highest	IM_FR (13020)	MI_FR (14088)	MI_FR (61.10%)	M_PR (40.00%)

Figure A.2: Benchmark Result By Domain (cont.)