Non-Prehensile Throwing: A Reinforcement Learning Perspective

***Abstract*— Robotic systems have adopted throwing skills to enable fast and far-reaching object transportation beyond traditional pick-and-place operations. However, prehensile (grasp-based) throwing limits the range of manipulable objects to those that are feasible to grasp. In contrast, achieving non-prehensile (grasp-free) throwing requires precise control of inertial and frictional forces. In this work, we propose learning a non-prehensile throwing policy via reinforcement learning. Our Markov Decision Process (MDP) formulation enables offline planning of throwing trajectories for random objects—including heavy (790 g) and large (20×20×28 cm) ones—to random target positions as far as 3.5 m and as high as 1.8 m, while operating at the robot's physical limits (5 m/s). Our simulation-trained policy is transferred to the real world in a zero-shot manner, achieving an average success rate of 85% across different objects and targets. Furthermore, our results demonstrate the policy's robustness to modeling uncertainties and its generalization to unseen objects, including cylindrical cans and partially-filled bottles.**

I. Introduction

Going beyond the limited workspace of quasi-static pick-and-place, throwing can expand the reachable workspace and enable faster object transportation. Consequently, many robotic systems have been developed to acquire this skill, which comes naturally to humans. For relatively small objects (e.g., a ball), throwing can be achieved by precisely timing the transition from a prehensile state (grasping) to a non-prehensile state (release via hand opening). For larger objects that cannot be grasped (e.g., a storage box or suitcase), one must rely on non-prehensile manipulation, where arm motion (using one or two hands) imparts the desired object pose and velocity before release through a rapid deceleration. Can robots do the same?

Many existing works [1, 2, 3] on prehensile throwing utilize a two-fingered gripper to control the object's release at a desired pose and velocity. While larger industrial robots with bigger grippers can handle larger objects [4], their performance degrades with smaller objects. Alternatively, bi-manual systems [5] can be employed; however, they may be excessive for handling small objects. Non-prehensile throwing has also been demonstrated in simplified setups [6, 7, 8], where inertial and frictional forces are controlled to throw a single object on a tilted table. Yet, robust and generalizable non-prehensile throwing remains an open challenge.

In this work, we propose a learning-based approach to develop a non-prehensile throwing policy, "NP-Throw." Using a tray-like end-effector, our policy is trained via reinforcement learning to generate throwing trajectories offline in just a few milliseconds. It can throw various objects—including large and heavy ones—to distant and elevated targets. Figure 1 shows our simulation-trained policy performing controlled sliding and throwing of different objects, successfully transferring to the real world. This performance is enabled by a carefully designed Markov Decision Process (MDP), accurate system identification of robot and object dynamics, and extensive analysis of policy behavior under unseen and uncertain conditions.

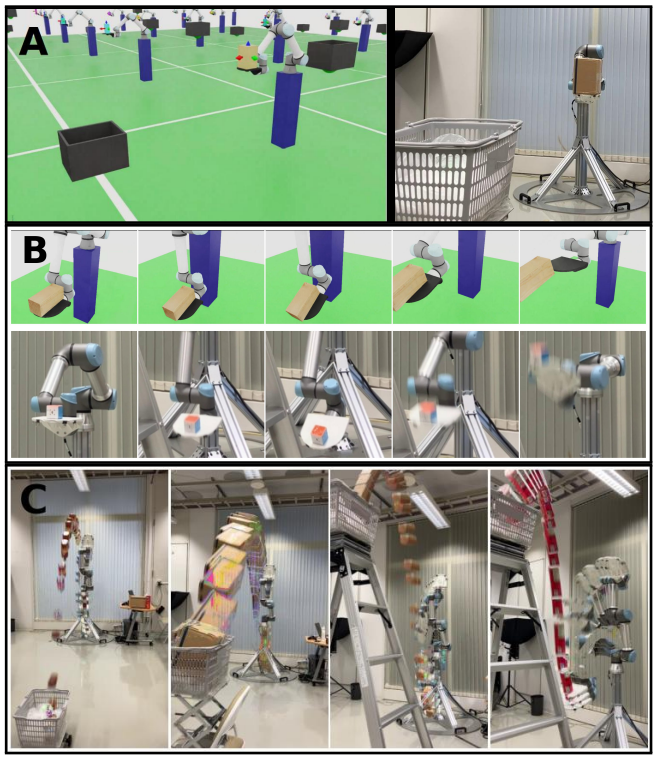

Fig. 1: Our **non-prehensile throwing** policy is **(A)** trained in simulation via reinforcement learning to throw randomized cuboids to a randomly placed target bin. Accurate throwing is enabled via **(B)** controlled sliding and throwing in both simulated and real-world setups. We demonstrate **(C)** successful throwing of objects including a partially-filled bottle, a large box, a heavy wood block, and a tall chips can. Videos are available at our project page: `https://tinyurl.com/np-throwing`

Our contributions are summarized as follows:

- A learning-based formulation for offline planning of non-prehensile throwing trajectories.
- Extensive simulation-based evaluation of the proposed method's performance, generalization, and robustness.
- Real-world validation demonstrating successful sim-to-real transfer across various objects and targets.
- Open-source code available at `https://tinyurl.com/np-throwing`.

II. Related Works

Prehensile Throwing using a two-fingered gripper has been adopted in many works. While trajectories can be optimized under the assumptions of projectile motion and instantaneous object release [5, 9], uncertainties arising from unmodeled drag forces [1], release timing [2], or the sim-to-real gap [3] must be compensated. To address these challenges, many learning-based approaches have been proposed. In [1], grasping and throwing networks were learned via self-supervision using residual physics to compensate for unmodeled forces in simple projectile scenarios. Reinforcement learning (RL) has been used to learn throwing policies either directly in the real world [10] or in simulation [11, 4, 12], followed by sim-to-real transfer. Sampling-based techniques have also been employed to generate datasets of throwing configurations and landing positions, which can be queried at inference time [3, 13]. These datasets have alternatively been used to train throwing policies via Decision Transformers [14]. In both [3, 14], a small number of real-world experiences are used to correct for the sim-to-real gap. Beyond static targets, throwing at moving targets has been explored in [12, 5]. In addition to fixed arms with two-fingered grippers, throwing policies have been developed for mobile manipulators [13], bi-manual systems [5], and dexterous hands [15].

Non-Prehensile Throwing is a specific application of non-prehensile manipulation [16], where target motion is achieved by controlling inertial and frictional forces. Model-based approaches have addressed non-prehensile throwing in simplified setups [6, 7, 8]. For example, on a 2D inclined table, a single object can be thrown to a target location by decelerating the end-effector at a specific release pose and velocity. In this work, we extend these ideas by applying learning-based frameworks (e.g., RL) to learn a generalized throwing policy capable of handling arbitrary objects and targets.

III. Method

We learn a non-prehensile throwing policy via reinforcement learning. First, we formulate the task as a **Markov Decision Process** (MDP) with carefully designed observation and action spaces, as well as a tailored reward function. To facilitate **sim-to-real transfer**, we adopt an acceleration-based action space and an up-sampling strategy to generate smooth joint velocity trajectories. After system identification of the robot dynamics, the simulation-trained policy is deployed zero-shot to the real setup using offline-generated trajectories.

A. Markov Decision Process Formulation

We design the observation space of the actor such that a state $\mathcal{S}$ is agnostic to the object state (i.e., pose or velocity), enabling offline trajectory planning. Starting from a given initial state $s_0 \in p(\mathcal{S}_0)$, an optimal action $a_t \in \mathcal{A}$ is selected by the policy to maximize the expected reward $\mathcal{R}$. The action a_t is processed into target joint velocities, which are then commanded to a lower-level controller. The reward function encourages timed and accurate throws toward the target bin. A careful choice of termination conditions and training hyper-parameters helps stabilize the learning behavior.

1) Initial State Distribution: In each episode, we randomize the object model, target position, and initial robot configuration using uniform distributions, with ranges listed in Table I. The object size (width, length, and height) is randomized based on prior knowledge of a target set of real objects listed in Table V, and the end-effector tray dimensions (20×20 cm). Similarly, we select ranges for mass, contact friction, and restitution to encompass the properties of the target objects. The target position ranges are roughly estimated by simulating different projectile trajectories, assuming a maximum end-effector velocity of 5 m/s—where our hardware, the UR5e, reaches its limit. The robot's initial configuration is randomized while ensuring that the tray's x-axis aligns with the world's positive x-axis (horizontal and facing forward). Alternative initializations (e.g., alignment with the world's negative x-axis) tend to encourage over-throw behavior [10], which we plan to address in future work. The object is always initialized at the center of the tray, and we assume that its center of mass (COM) aligns with its geometric center.

TABLE I: Sampling ranges used for randomizing the object model, target position, and initial joint configuration of the robot.

Property	Unit	Distribution
Object Model		
Object Width/Length	cm	$\mathcal{U}(5, 20)$
Object Height	cm	$\mathcal{U}(5, 25)$
Object Mass	kg	$\mathcal{U}(0.1, 1.0)$
Object/Tray Static Friction	-	$\mathcal{U}(0.2, 1.0)$
Object/Tray Dynamic Friction	-	$\mathcal{U}(0.2, 1.0)$
Object/Tray Restitution	-	$\mathcal{U}(0.0, 0.5)$
Target Position		
Target Distance	cm	$\mathcal{U}(150, 350))$
Target Elevation	cm	$\mathcal{U}(0, 200))$
Initial Joint Configuration		
Joint 2 Position q_2	rad	$-\pi/4 \pm \mathcal{U}(-\pi/4, \pi/4)$
Joint 4 Position q_4	rad	$-\pi/4 \pm \mathcal{U}(-\pi/4, \pi/4)$
Joint 3 Position q_3[a]	rad	$q_3 = -(q_2 + q_4)$

[a] Ensuring a horizontal tray at initialization.

2) Observation Space: To enable offline trajectory planning, the actor's observation state is designed to be recursively updated based on the latest policy action, as illustrated in Fig. 2. Successful learning is achieved under two key assumptions. First, the environment is deterministic, such that the joint state trajectory (position and velocity) and object model can implicitly describe the object state trajectory in the absence of external disturbances. Second, the low-level controller can accurately track the commanded joint velocities (or positions) in both simulated and real setups, ensuring that recursively generated joint state trajectories match the ground-truth trajectories. This approach improves both policy learning and deployment by mitigating simulator stochasticity (as seen in GPU-powered IsaacSim) and real-world sensor noise and latency.

Accordingly, we design the observation space as listed in Table II. The object model (size and physical properties) and target position are fixed for a given trajectory. A history of length H—including joint positions, velocities, and policy actions—is updated at every step. To improve learning

behavior and policy performance, we adopt an asymmetric actor–critic architecture, where the critic receives privileged information, including the end-effector and object states (pose and velocity). Additionally, a binary state indicating whether the object has been thrown (i.e., zero contact force between the object and the tray) is included. This state, i_{thrown_t}, is used in the reward formulation and is important for accurate value function estimation.

TABLE II: Observation space for the asymmetric actor–critic architecture. Joint position, velocity, and action histories of length H are used to address potential partial observability.

Input State	Symbol	Dim.	Actor	Critic
Action History	$a_{t-H:t}$	3H	✔	✔
Target Position	p_{target}	2	✔	✔
Object Model				
Object Size	$[w, l, h]_O$	3	✔	✔
Object mass	m_O	1	✔	✔
Object/Tray Friction	$[\mu_s, \mu_k]$	2	✔	✔
Object Restitution	e	1	✔	✔
Robot State				
Joint Position	$q_{t-H:t}$	3H	✔	✔
Joint Velocity	$\dot{q}_{t-H:t}$	3H	✔	✔
End-Effector Position	p_{EE_t}	3	✗	✔
End-Effector Rot. Mat.	R_{EE_t}	6	✗	✔
End-Effector Velocity	v_{EE_t}	6	✗	✔
Object State				
Object Position	p_{O_t}	3	✗	✔
Object Rot. Mat.	R_{O_t}	6	✗	✔
Object Velocity	v_{O_t}	6	✗	✔
Object Thrown (Binary)	i_{thrown_t}	1	✗	✔

✔ and ✗ refers to state presence or absence, respectively.

3) Action Space: With trajectory smoothness as a core selection criterion, the control frequency on the physical system is set to the maximum supported rate (500 Hz for the UR5e). However, running simulations at this rate can be prohibitively slow. Therefore, we opt for a lower simulation rate (120 Hz) and an even lower control rate (20 Hz) to avoid excessively long training runs caused by extended horizon lengths. To achieve smooth up-sampling from 20 Hz to 120/480 Hz, we adopt an acceleration-based action space, where a constant acceleration is commanded for N steps with $dt = 50/N$ ms. At each step, joint positions and velocities are updated recursively following (1). Since we integrate over a fixed time window of 50 ms per action step, different up-sampling rates result in matching terminal position and velocity states, enabling seamless transfer between simulated and real setups.

For safe deployment on the physical system, we interpret the policy output as an acceleration rate (jerk) and limit it to 4 m/s^2 per step to reduce jerky robot motion. Additionally, we estimate joint velocity for a given acceleration and clip the acceleration to avoid exceeding the joint speed limit (3 rad/s), using the relation $\hat{\ddot{q}} = \frac{\dot{q}_{\max} - \dot{q}_t}{dt}$. After an object is thrown, we dampen the robot's velocity by applying a Proportional-Derivative (PD) controller with empirically tuned gains: $\hat{\ddot{q}} = -0.5\frac{\dot{q}_t}{dt}$. Encouraging this behavior by penalizing the joint velocity norm during training resulted in suboptimal policy performance, which may be attributed to poor gradient flow—since robot actions have limited influence on task success in the post-throw phase.

$$q_t = q_{t-1} + \dot{q}_{t-1}dt + \frac{1}{2}\ddot{q}_t dt^2 \quad , \quad \dot{q}_t = \dot{q}_{t-1} + \ddot{q}_t dt \qquad (1)$$

4) Reward Formulation: Table III lists a set of reward terms designed to encourage fast and accurate throwing, including live, dense, sparse, and regularization rewards. The total reward is a weighted sum of individual components, with weights tuned only roughly.

Live-Thrown Reward. During early training phases, this term dominates, encouraging longer episode durations (i_{live}) and fast object throwing (i_{thrown}). Using only i_{live} led to unnecessary holding of the object.

Double-Throw Penalty. After observing reward hacking—where the policy repeatedly performed in-place throwing and catching—a double-throw penalty (i_{DT}) was introduced to discourage contact with the object after it has been thrown.

Position Error Penalty. This dense reward is crucial for guiding initial exploration, which fails if only sparse rewards are used. The position error is computed as the L2 norm between the object position p_{O_t} and the target position p_{target}.

Velocity towards Target. To encourage movement toward the target, we reward based on the dot product between the normalized object velocity vector v_{O_t} and the vector from the current object position p_{O_t} to the target position p_{target}. This term also incentivizes direct throws, avoiding unnecessarily high projectiles.

Sparse Success Reward. This term serves as the main objective for the bin-throwing task. The binary success term i_{Success} indicates whether the object is within a threshold distance (25 cm, half the real bin length) from the target position. Accurate throwing is further encouraged by incorporating a continuous kernel function $\mathcal{K}(r_p)$, similar to [17].

Regularization Penalties. To achieve smooth and efficient acceleration trajectories, we penalize the norm of joint accelerations $\ddot{q}_t$ and the change in acceleration $(\ddot{q}t - \ddot{q}t - 1)$. To dampen joint velocity during the post-throw phase, we heavily penalize joint velocities $\dot{q}t$. These penalties are scaled proportionally to the average success reward rSuccess to avoid excessive regularization during early training phases.

TABLE III: Reward formulation and corresponding weights for live, dense, sparse, and regularization components.

Reward Term	Symbol	Formulation	Weight
Live-Thrown	r_L	$i_{live} \times i_{thrown}$	10
Double Throw	r_{DT}	i_{DT}	-100
Position Error	r_p	$\|\|p_{O_t} - p_{target}\|\|$	-1
Velocity to Target	$r_{v_{goal}}$	$\langle \frac{v_{O_t}}{\|v_{O_t}\|}, \frac{p_{O_t} - p_{target}}{\|p_{O_t} - p_{target}\|} \rangle$	10
Success	$r_{Success}$	$i_{Success}\mathcal{K}(r_p)$	100
Joint Acc. Reg.	$r_{\ddot{q}}$	$\|\|\ddot{q}_t\|\|$	-0.1
Joint Jerk Reg.	$r_{\dddot{q}}$	$\|\|\ddot{q}_t - \ddot{q}_{t-1}\|\|$	-0.1
Joint Vel. Reg.	$r_{\dot{q}}$	$\|\|\dot{q}_t\|\| \times i_{thrown}$	-10

5) Episode Termination: The episode is terminated early upon failure to prevent low-quality experiences from being added to the training buffer. Failure conditions include the

object falling to the ground ($p^z_{O_t} < 0$) and excessive backward motion of the end-effector ($p^x_{EE_t} < -0.5$). A complete episode lasts for 3.2 seconds, corresponding to 64 steps under a 20 Hz control rate.

6) Implementation Details: We adopted the NVIDIA IsaacLab framework [18] to efficiently parallelize data collection in the NVIDIA Isaac Sim simulator and policy learning using the RL-Games [19] implementation of Proximal Policy Optimization (PPO) [20]. The policy outputs actions at 20 Hz, which are up-sampled to 120 Hz in simulation and 480 Hz in the real environment. Setting the horizon length equal to the episode length (64 steps) improves learning stability by optimizing over the full episode duration. Additional PPO hyper-parameters can be found in our released code at `https://tinyurl.com/np-throwing`. Training the policy across 4096 environments for 1000 epochs (over 250 million steps) took just over an hour on a single NVIDIA RTX 4090 GPU. Our physical system uses a Universal Robots UR5e, which has a maximum reach of under one meter and a maximum end-effector velocity of 5 m/s. The system is controlled via ur-rtde [21], a Real-Time Data Exchange (RTDE) wrapper for UR robots.

B. Sim-to-Real Transfer

The simulation-trained policy can be successfully transferred to real-world setups by minimizing the sim-to-real gap in both robot and object dynamics through **system identification**. Additionally, in this work, the policy is deployed as an **offline motion planner** in an open-loop manner to mitigate challenges posed by sensor noise and control latency.

1) System Identification: For robot dynamics, we perform system identification by executing a set of minimum-jerk cyclic trajectories with varying amplitudes and frequencies on the physical system, and recording its joint position and velocity trajectories. In simulation, these trajectories are executed across 4096 parallel environments with varying actuator stiffness and damping gains. The optimal gains are those that minimize a weighted sum of joint position and velocity errors. Further refinement is performed by sampling additional populations around the optimized gains. Empirically, we found that controlling the robot via low-level velocity commands resulted in a smaller sim-to-real gap compared to position control. To avoid extensive modeling of object dynamics, we assume policy robustness to variations in object size, mass, center of mass, and placement (see next section). Therefore, only the contact friction parameters (static and dynamic coefficients) are manually tuned around nominal values to maximize real-world success rates. This tuning was performed for a single object (Rubik's cube), and the same friction values were used for all objects. Nonetheless, online identification of such physical properties could further enhance policy performance.

2) Trajectory Planning & Deployment: As shown in Fig. 2, trajectory planning is performed offline without accessing ground-truth states in either simulation or real-world setups. Given the object model, target position, and initial joint configuration, joint positions and velocities are recursively updated using (1), based on the computed acceleration command. The 20 Hz acceleration trajectory is up-sampled to velocity trajectories for deployment at 120 Hz in simulation and 480 Hz in the real environment.

During training, the policy has access to the binary state i_{thrown}, which enables velocity damping via PD control in the post-throw phase. However, this state is not accessible during deployment. While contact sensors in simulation or force/torque sensors in real setups could provide this information, doing so would require closed-loop feedback at every timestep. As an alternative, the throwing time can be estimated based on the object model, target position, and initial robot configuration (see Table V).

To this end, we trained a simple regression model using the random forest algorithm from the *Scikit-learn* library [22]. The training data consists of 4096 successful trajectories generated by the trained policy. The regressor predicts the throwing step with a mean squared error (MSE) of 2.12 ± 1.65 steps. A conservative version of the regressor shifts the predicted throwing step by three steps to avoid premature application of PD damping, resulting in fully offline trajectories with only a marginal drop in success rates.

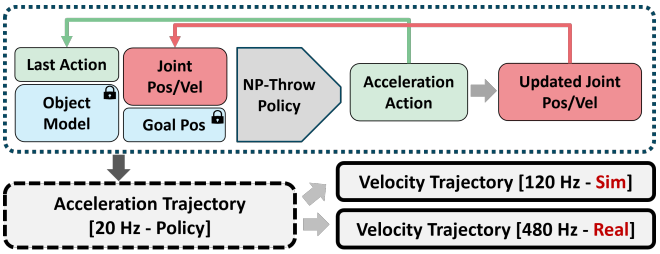

Fig. 2: Throwing trajectory planning using the "NP-Throw" policy. Given an object model, target position, and initial joint configuration, the acceleration trajectory is generated offline by recursively computing policy actions and updating joint positions and velocities. The trajectory is deployed in both simulated and real setups by generating up-sampled velocity trajectories, following (1).

IV. Evaluation

We evaluate the performance of the "NP-Throw" policy in both simulation and on a physical system, addressing the following key questions:

1) **Ablations**: How does the proposed formulation compare to ablated design choices?
2) **Performance**: How does the trained policy perform across different object models and target positions?
3) **Generalization**: Does the policy generalize to unseen object models and target positions?
4) **Robustness**: How robust is the policy to uncertainties in object modeling?
5) **Real-World**: How well does the policy perform on a physical system with various objects and target positions?

A. Ablation Studies

We ablate different design choices by evaluating 4096 configurations—comprising object models, target positions, and initial joint positions—under four random seeds. Figure

3 shows the success rate across different models. In terms of reward formulation, the policy failed to learn accurate throwing behavior when guided solely by dense rewards (r_p and $r_{v_{\text{target}}}$), resulting in a zero success rate. Less accurate throws were also observed when either the live or sparse reward terms were omitted, due to the absence of their dominant reward contributions. Regarding action formulation, limiting the control degrees of freedom (DOF) to 3 was sufficient for planar throwing. Using 6 DOF (referred to as dexterous in [2]) degraded policy performance and led to unsafe motions unsuitable for real-world deployment. Additionally, explicitly learning the velocity damping behavior in the post-throw phase resulted in a performance drop, likely due to poor learning signals after the object is thrown. We further ablate the policy control rate and maximum acceleration (jerk), showing a drop in performance when deviating from the proposed combination (20 Hz and 4 m/s²). Lower rates (e.g., 10 Hz and 2 m/s²) failed to achieve accurate wide-range throws, while higher rates led to excessive end-effector motion, causing early termination and hindering stable learning. Finally, we examine the effect of observation history length. Both no history and short histories yielded comparable performance, whereas longer histories unnecessarily increased the policy input size with redundant information.

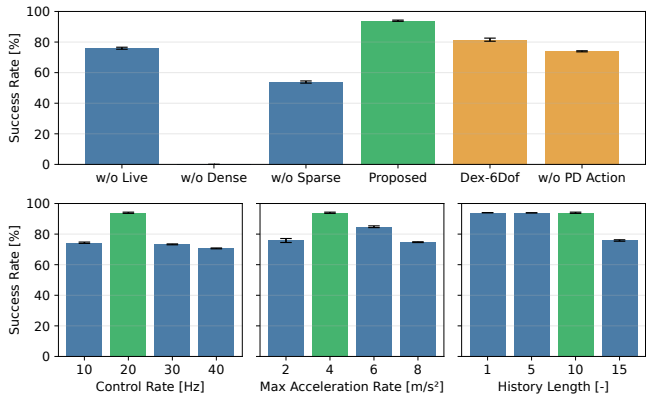

Fig. 3: Ablation study of different design choices and their impact on success rates. A combination of live, dense, and sparse rewards was crucial for stable learning. For planar throwing, dexterous control of all six joints led to a drop in performance. Policy control rate, maximum acceleration (jerk), and observation history length were also ablated, with the green-colored bar indicating our selected configuration.

B. Performance Analysis

We evaluate the policy performance by running 4096 configurations across 10 random seeds and analyzing the results using 2D histograms, as shown in Fig. 4. For target positions seen during training, the policy maintains high throwing accuracy across most distance and elevation combinations, except for relatively far and high targets (greater than 1.5 m in height and 3 m in distance). While the policy can reach far and low targets, higher targets at most distances remain more challenging. Regarding object size, the policy underperforms for relatively tall and thin objects, which are harder to balance and tend to rotate excessively when thrown toward distant targets. In contrast, longer objects (along x-axis) tend to stabilize sliding and throwing behavior, resulting in higher success rates. As for the object model, mass and restitution coefficients had no notable impact on performance. However, friction coefficients showed more subtle effects. High static friction increased object sticking, inducing excessive rotation—especially for tall objects. Lower dynamic friction (which behaves similarly to rolling friction in IsaacSim) mitigated this behavior and improved success rates. These findings suggest that surfaces with lower static and dynamic friction are more suitable for non-prehensile throwing. This was validated empirically: a rubber surface ($\mu = 0.8$) yielded poor real-world performance compared to a paper surface ($\mu = 0.4$). Finally, unless initialized at extreme joint configurations (e.g., near singularities), the policy demonstrated consistently good performance across different initial configurations.

C. Policy Generalization

While the policy performs best for configurations within the training distributions (Table I), we evaluate its generalization to unseen distributions and arbitrary objects, as listed in Table IV. When evaluating cylindrical objects, only a minor drop in performance was observed, indicating that the policy primarily relies on surface contact during sliding and throwing, rather than edge interactions—which behave differently in the case of cylinders. When varying the object model, the policy performed well for low ranges of object mass, friction, and height. However, higher mass and friction values induced stronger frictional forces, resulting in poor sliding due to sticking. Taller objects also showed reduced performance due to their higher height-to-length ratio and increased likelihood of colliding with the bin. The throwing range is mostly limited by the physical specifications of the robot. In attempts to reach higher targets, success was feasible up to 2.2 m in elevation for targets around 2 m in distance. For farther targets, reasonable success was achieved up to 3.8 m in distance for elevations up to 0.8 m. While some success was observed at higher and farther targets, it was limited to specific object sizes and physical properties. Additionally, we evaluate the success rate for a selection of YCB [23] objects, where high performance was demonstrated for most cases. Among cylindrical objects, those with a larger base area (e.g., "pitcher") performed better than those with a smaller base (e.g., "chips"). For objects with a high width-to-length ratio, performance improved when the object was rotated (-Rot) to align its longer edge with the x-axis. The policy also successfully handled non-uniform objects such as the "power drill," which features a shifted center of mass.

D. Policy Robustness to Modeling Uncertainty

Prior to deploying the policy in the real world, we evaluate its robustness to modeling uncertainties, aiming to reduce the burden of precise object modeling. This sensitivity analysis is conducted in simulation by perturbing the policy input away from the ground-truth object model, as shown in Fig. 5. Our base object is a $10\times10\times10$ cm cube with a mass of 0.5 kg

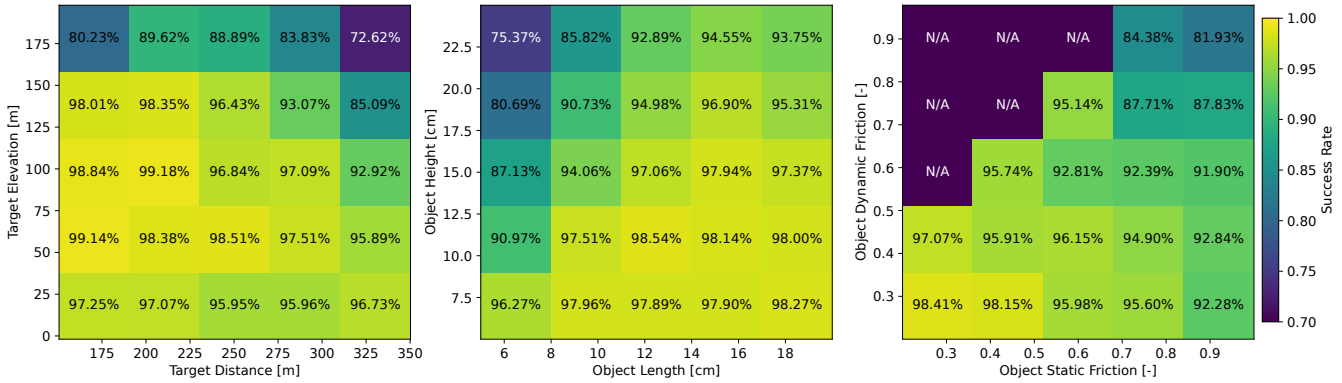

Fig. 4: 2D histograms analyzing the performance of the "NP-Throw" policy in terms of target position, object size, and contact friction. The policy fails more frequently for high (above 150 cm) and far (beyond 300 cm) targets. Objects with a high height-to-length ratio are more difficult to balance and throw. High static and dynamic friction limits object sliding, resulting in reduced performance. Entries where the dynamic friction is higher than static friction are not applicable (N/A).

TABLE IV: Evaluation of policy generalization to unseen objects, including cylindrical shapes, novel physical properties, and a selection of YCB objects [23]. Throwing accuracy degrades under extreme increases in object mass, friction, target distance, and elevation. However, the policy maintains high performance for generic everyday objects such as those in the YCB dataset. Success rates below 50% are highlighted in red.

Object	S.R. [%]	Object	S.R. [%]
Cuboids [Training]	93.9	Cylinders	93.1
Object Model			
Mass [0.01-0.1] kg	93.6	Mass [1-5] kg	57.1
Friction [0.01-0.2]	83.3	Friction [1-10]	28.2
Height [1-5] cm	97.0	Height [25-40] cm	68.4
Target Position			
Elevation [2-2.5] m	40.7	Elevation [2.5-3] m	1.3
Distance [3.5-4] m	53.7	Distance [4-4.5] m	3.0
YCB Objects			
WoodBlock	99.1	Chips	87.6
Spam	99.4	Pitcher	95.9
CrackerBox	87.6	CrackerBox-Rot	99.4
MustardBottle	63.5	MustardBottle-Rot	93.5
BleachCleanser	68.0	BleachCleanser-Rot	98.5
PowerDrill	65.9	PowerDrill-Rot	98.4

and static/dynamic friction coefficients of 0.5. The object is evaluated under 512 configurations of target positions and initial joint configurations. When perturbing the mass, the policy maintains near-perfect success rates, demonstrating high robustness to mass uncertainty. This may be explained by the dual relationship between inertial and frictional forces, which we leave as a subject for future work. Similarly, we observe limited sensitivity to the coefficient of restitution. The policy is most sensitive to inaccuracies in friction modeling, where the object either slides excessively or sticks to the tray. In both cases, the throwing timing is disrupted, resulting in trajectory failure. These results suggest that careful tuning of friction parameters is crucial for successful deployment. Regarding object size, moderate uncertainty can be tolerated with only minor drops in performance. Underestimating object height and overestimating its length tends to produce more robust trajectories. For example, a trajectory optimized for a long, flat object (e.g., a horizontal wood block) can be successfully executed for various objects with different masses, assuming similar frictional contact. This "Throw-Anything" behavior was demonstrated on the physical system,

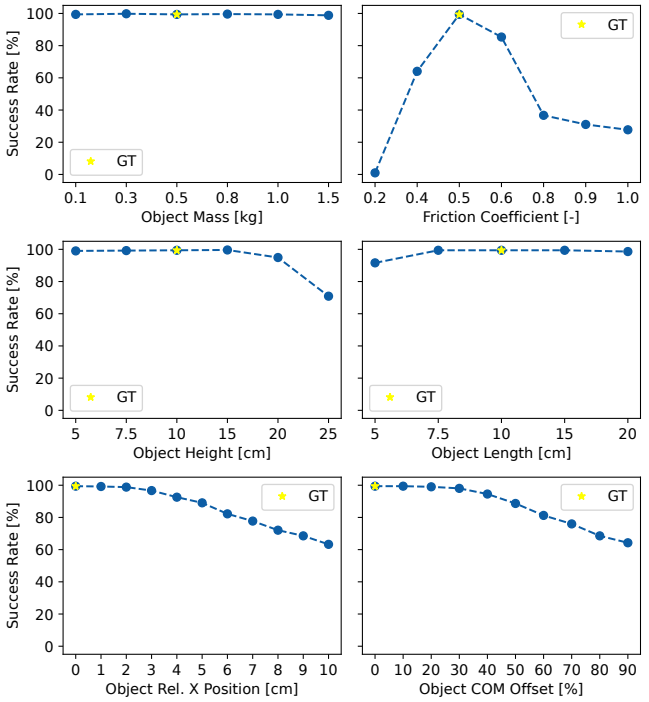

Fig. 5: Evaluation of policy sensitivity to modeling uncertainties, measured in terms of success rate. By varying the input object model from the ground-truth configuration (yellow star), the policy demonstrates robustness to mass and size uncertainty, but high sensitivity to friction inaccuracies. Off-center object placement on the tray or shifts in the object's center of mass (COM) lead to a gradual decline in performance proportional to the offset or shift.

as shown in `https://tinyurl.com/np-throwing`. Off-center object placement on the tray results in offset release and trajectory deviation. Increasing this offset gradually reduces success rates, especially for distant targets. Similarly, shifts in the center of mass (COM) can degrade policy performance. However, this effect is largely attributed to inaccuracies in IsaacLab's randomization functions [18], as both simulated (e.g., "power drill") and real-world (e.g., "liquid bottle") experiments demonstrated successful throws despite COM shifts.

E. Real-World Evaluation

For real-world evaluation, we selected five objects with distinct physical properties: a small Rubik's cube, a heavy wood block (790 g), a tall and thin chips can, a liquid bottle with a shifting center of mass (COM), and a large box. Target positions included three distances (150 cm, 250 cm, and 350 cm) and three elevations (15 cm, 80 cm, and 180 cm). For each of the 45 combinations ($5\times3\times3$), the policy was evaluated in simulation across 512 different initial joint configurations, and only 10 successful trajectories were deployed on the physical system. To reduce friction, a sheet of paper was placed on top of the rubber tray. Friction parameters were tuned to $\mu_s = 0.35$ and $\mu_k = 0.3$ by maximizing the Rubik's cube success rate at the (250 cm, 80 cm) target. These values were then applied uniformly across all objects.

Results listed in Table V show an average success rate of 85% across the 45 combinations. Most failures occurred at the (350 cm, 180 cm) target, likely due to hardware limitations—specifically, the inability of the real system to reach the required end-effector velocity exceeding 5 m/s. At distant targets, trajectory deviations caused by modeling inaccuracies (e.g., friction, placement) are amplified. For the large box, failures at the (150 cm, 180 cm) target were often due to limited task space, resulting in object–bin collisions. Trajectory statistics listed in Table VI revealed correlations between target position and both the maximum end-effector (EE) velocity and maximum object height. The EE pitch angle increases for distant targets but remains flatter for higher elevations (e.g., 150 cm and 250 cm). Throwing and reaching times also correlate with target position, with taller and more distant targets requiring later throw and reach times. These correlations can be leveraged to estimate the throwing time during offline planning, as discussed in Section III-B.2.

V. Conclusion

We proposed a learning-based non-prehensile throwing policy for arbitrary objects aimed at random target positions. In simulation, we extensively evaluated the policy's performance, generalization, and robustness, highlighting the importance of tuning contact friction and minimizing the object's height-to-length ratio for successful throws. Offline-generated trajectories were successfully deployed in the real world, achieving an average success rate of 85% across a diverse set of challenging objects and target positions.

Limitations & Future Work. This work focused on planar throwing toward forward-facing targets. Expanding the workspace to full 3D space [4] and incorporating additional control degrees of freedom [2] could enable more efficient and flexible throwing strategies, especially in environments with obstacles. Rather than relying on manual and often imprecise object modeling—particularly for friction—autonomous dynamics modeling could be achieved by executing predefined trajectories that induce sliding or rotation, and matching observed object states (via vision, force, or acoustic feedback) to simulated or ground-truth models. Throwing heavy objects to distant targets may result in high-impact collisions. Developing impact-aware throwing strategies and learning a complementary catching policy [8, 24] would allow for safer deployment. Extending this work to practical applications such as material handling and automated sorting would also be a promising direction.

References

[1] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," 2019.

[2] Y. Liu and A. Billard, "Tube acceleration: Robust dexterous throwing against release uncertainty," *IEEE Transactions on Robotics*, vol. 40, pp. 2831–2849, 2024.

[3] S. Kim, A. Coninx, and S. Doncieux, "From exploration to control: Learning object manipulation skills through novelty search and local adaptation," *Robotics and Autonomous Systems*, vol. 136, p. 103710, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889020305509

[4] L. Werner, F. Nan, P. Eyschen, F. A. Spinelli, H. Yang, and M. Hutter, "Dynamic throwing with robotic material handling machines," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024. [Online]. Available: https://arxiv.org/abs/2405.19001

[5] M. Bombile and A. Billard, "Bimanual dynamic grabbing and tossing of objects onto a moving target," *Robotics and Autonomous Systems*, vol. 167, p. 104481, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889023001203

[6] J. Z. Woodruff and K. M. Lynch, "Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4066–4073.

[7] A. Pekarovskiy and M. Buss, "Optimal control goal manifolds for planar nonprehensile throwing," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 4518–4524.

[8] A. Pekarovskiy, F. Stockmann, M. Okada, and M. Buss, "Hierarchical robustness approach for nonprehensile catching of rigid objects," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3649–3654.

[9] F. Lombai and G. Szederkenyi, "Throwing motion generation using nonlinear optimization on a 6-degree-of-freedom robot manipulator," in *2009 IEEE International Conference on Mechatronics*, 2009, pp. 1–6.

[10] S. Aslam, K. Kumar, P. Zhou, H. Yu, M. Yu Wang, and Y. She, "Dartbot: Overhand throwing of deformable objects with tactile sensing and reinforcement learning," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 13 644–13 661, 2025.

[11] Y. Ma, Y. Liu, K. Qu, and M. Hutter, "Learning accurate whole-body throwing with high-frequency residual policy and pullback tube acceleration," in *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2025. [Online]. Available: https://arxiv.org/abs/2506.16986

[12] H. Kasaei and M. Kasaei, "Throwing objects into a moving basket while avoiding obstacles," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 3051–3057.

[13] Y. Liu, A. Nayak, and A. Billard, "A solution to adaptive mobile manipulator throwing," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 1625–1632.

[14] M. Monastirsky, O. Azulay, and A. Sintov, "Learning to throw with a handful of samples using decision transformers," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 576–583, 2023.

[15] A. Petrenko, A. Allshire, G. State, A. Handa, and V. Makoviychuk, "Dexpbt: Scaling up dexterous manipulation for hand-

TABLE V: Real-world evaluation of the "NP-Throw" policy across objects with varying properties. The notably high success rate in simulation partially transfers to the real world, with an average success rate of **85%**. Failures were more common for far and high targets due to physical limitations of the robot. Close and high targets were also challenging for large objects due to limited task space.

	Rubik's Cube 6×6×6 cm		**Wood Block** 20×8.5×8.5 cm		**Chips** 7.5×7.5×25 cm		**Liquid Bottle** 6×6×17 cm		**Large Box** 20×20×28 cm	
										
Distance [cm]	**Elevation [cm]**	**SR Sim [%]**	**SR Real [/10]**	**SR Sim [%]**	**SR Real [/10]**	**SR Sim [%]**	**SR Real [/10]**	**SR Sim [%]**	**SR Real [/10]**	**SR Sim [%]**	**SR Real [/10]**
150	15	100	10	100	10	100	9	99	10	95	10
	80	100	9	100	8	98	10	100	10	100	10
	180	82	10	90	10	63	6	84	9	74	4
250	15	98	8	100	8	99	8	99	10	100	9
	80	100	9	100	8	99	8	100	10	99	9
	180	99	8	100	9	98	7	97	9	99	7
350	15	97	9	99	10	99	7	97	9	96	10
	80	100	10	99	10	88	10	55	9	94	9
	180	88	5	100	7	99	7	72	6	78	4

TABLE VI: Aggregated statistics of throwing trajectories across different objects and target positions. Maximum end-effector (EE) velocity, object height, EE pitch angle, and throwing/reaching times were found to correlate with target distance and elevation.

Distance	**Elevation**	**Max Vel.**	**Max Height**	**Max Pitch**	**Throw Time**	**Reach Time**	**Success Rate**	
[cm]	[cm]	[m/s]	[m]	[Degree]	[s]	[s]	**Sim** [%]	**Real** [/10]
150	15	1.45 ± 0.25	1.11 ± 0.1	34.03 ± 3.69	0.54 ± 0.05	0.93 ± 0.06	98.83	9.8
	80	2.26 ± 0.51	1.34 ± 0.09	27.51 ± 2.42	0.56 ± 0.05	0.95 ± 0.07	99.57	9.4
	180	4.02 ± 0.03	2.3 ± 0.06	20.14 ± 0.88	0.78 ± 0.09	1.48 ± 0.16	78.87	7.8
250	15	3.78 ± 0.08	1.6 ± 0.07	36.65 ± 0.96	0.7 ± 0.09	1.5 ± 0.14	99.18	8.6
	80	3.92 ± 0.05	1.71 ± 0.07	35.87 ± 1.07	0.81 ± 0.08	1.52 ± 0.13	99.41	8.8
	180	4.83 ± 0.06	2.37 ± 0.09	27.63 ± 0.62	1.06 ± 0.07	1.84 ± 0.11	98.63	8.0
350	15	4.20 ± 0.04	1.64 ± 0.1	36.6 ± 1.5	1.01 ± 0.07	1.91 ± 0.11	97.89	9.0
	80	4.78 ± 0.06	1.9 ± 0.11	36.56 ± 0.88	1.08 ± 0.07	1.96 ± 0.12	87.3	9.6
	180	5.15 ± 0.04	2.64 ± 0.08	42.75 ± 0.55	1.21 ± 0.06	2.2 ± 0.13	87.34	5.8

arm systems with population based training," in *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[16] F. Ruggiero, V. Lippiello, and B. Siciliano, "Nonprehensile dynamic manipulation: A survey," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1711–1718, 2018.

[17] A. Allshire, M. MittaI, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, and A. Garg, "Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2022. [Online]. Available: http://dx.doi.org/10.1109/IROS47612.2022.9981458

[18] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.

[19] D. Makoviichuk and V. Makoviychuk, "Rl games," 2021. [Online]. Available: https://github.com/Denys88/rl_games/

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[21] A. P. Lindvig, I. Iturrate, U. Kindler, and C. Sloth, "ur-rtde: An interface for controlling universal robots (ur) using the real-time data exchange(rtde)," in *2025 IEEE/SICE International Symposium on System Integration (SII)*, 2025, pp. 1118–1123.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics Automation Magazine*, no. 3, 2015.

[24] S. Kim, A. Shukla, and A. Billard, "Catching objects in flight," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1049–1065, 2014.