Cherry-Picking with Reinforcement Learning: Robust Dynamic Grasping in Unstable Conditions

Yunchu Zhang*†, Liyiming Ke*‡, Abhay Deshpande‡, Abhishek Gupta‡ and Siddhartha Srinivasa‡
*equal contribution
†The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA
‡Paul G Allen School of Computer Science and Engineering, Seattle, USA

Abstract*—Grasping small objects surrounded by unstable or non-rigid material plays a crucial role in applications such as surgery, harvesting, construction, disaster recovery, and assisted feeding. This task is especially difficult when fine manipulation is required in the presence of sensor noise and perception errors; errors inevitably trigger dynamic motion, which is challenging to model precisely. Circumventing the difficulty to build accurate models for contacts and dynamics, data-driven methods like reinforcement learning (RL) can optimize task performance via trial and error, reducing the need for accurate models of contacts and dynamics. Applying RL methods to real robots, however, has been hindered by factors such as prohibitively high sample complexity or the high training infrastructure cost for providing resets on hardware. This work presents CherryBot, an RL system that uses chopsticks for fine manipulation that surpasses human reactiveness for some dynamic grasping tasks. By integrating imprecise simulators, suboptimal demonstrations and external state estimation, we study how to make a real-world robot learning system sample efficient and general while reducing the human effort required for supervision. Our system shows continual improvement through 30 minutes of real-world interaction: through reactive retry, it achieves an almost 100% success rate on the demanding task of using chopsticks to grasp small objects swinging in the air. We demonstrate the reactiveness, robustness and generalizability of CherryBot to varying object shapes and dynamics (e.g., external disturbances like wind and human perturbations). Videos are available at https://goodcherrybot.github.io/

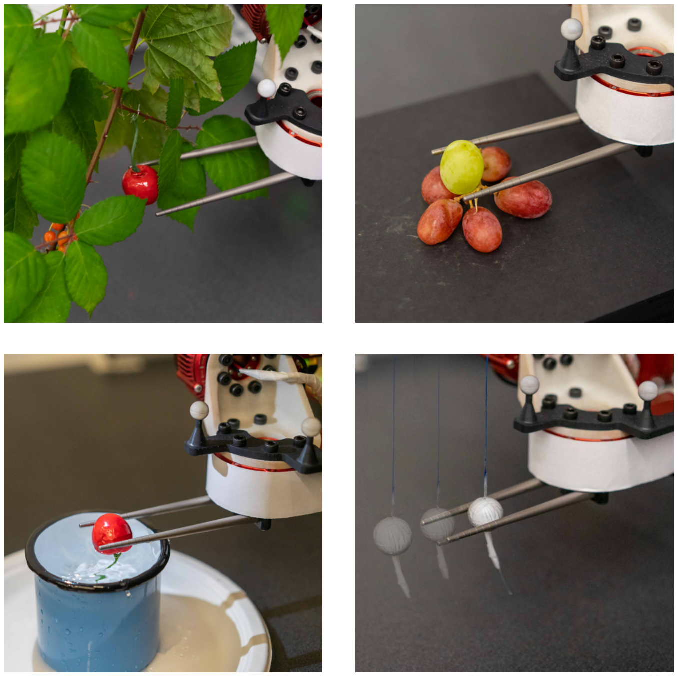


Fig. 1: The CherryBot system generalizes to various scenarios of dynamic fine manipulation: blowing wind, sliding grape cluster, moving water, and swinging string.

I. Introduction

How can we automate the task of picking cherries from a tree branch that is blowing in the wind, causing the branch to shake and the cherries to tremble? This scenario is an example of fine grasping *without rigid-surface support*, and its challenges are two-fold. First, for fine manipulation of small objects, perception errors and sensor noise dominate, making it difficult to grasp the objects precisely [1, 2]. Second, the problem is inherently dynamic since any contact with the object might set the entire scene into motion, which is complicated to model [3, 4]. Similar challenges arise in our everyday interactions, from mundane tasks such as removing broken shells from gelatinous egg whites to surgical tasks that detach clots from deformable organs. Given the ubiquitous nature of these tasks, developing robotic solutions to automate them holds immense practical and economic value.

For a predetermined, specific task, it is possible to invest in dedicated hardware [5, 6], specialized tools [7, 8, 9], and elaborately designed systems [10, 11] to solve these challenges. However, this research investigates a more universal solution: assuming that fine manipulation is required, inaccuracy is unavoidable and real-time reaction is necessary, can we enable dynamic fine grasping without stable support? An ideal agent should be:

- **Precise** enough to increase the likelihood of task success.
- **Robust** to perception errors and sensor noises that are likely to arise in the fine manipulation domain.
- **Reactive** to hard-to-model dynamic scenarios, external perturbations, and changes caused by its own movements.
- **Generalizable** to objects with different sizes, shapes and textures.

To address these challenges, we build a test bed using generic hardware that includes a robot arm and chopsticks for fine manipulation. The design of chopsticks is simple yet versatile and has been widely adopted in surgery [12, 13], meal assistance [14] and micro-manipulation [2, 15, 16]. We note that the thin rods of chopsticks make fine manipulation more difficult and that our assembled hardware has sensing and

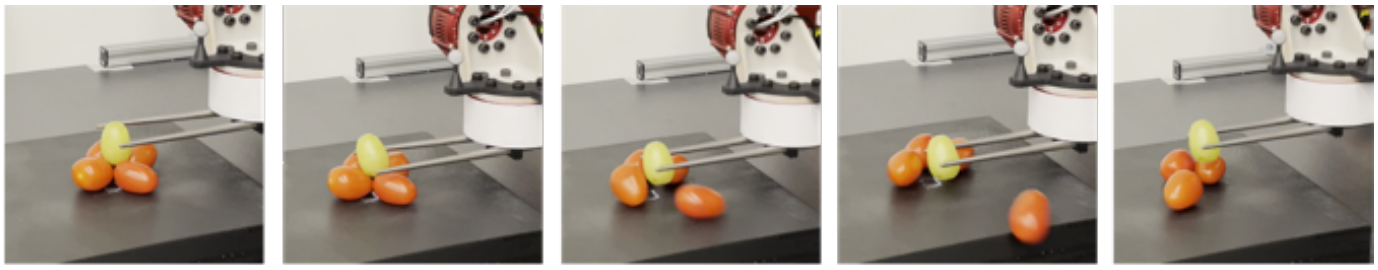


Fig. 2: The dynamic grasping task is challenging: any contact with an object might set it into motion, which is difficult to model. This challenge is exacerbated when fine manipulation is required, especially in the presence of sensor noise and perception errors.

actuation inaccuracies. However, the insights drawn from our accessible setup could be easily transferred to other platforms or tools that are built to operate with higher precision [5, 17, 18].

Prior work constructed analytic models [19, 20, 21] or motion primitives [22] for manipulation tasks with rich contacts. Instead, we choose reinforcement learning (RL) to circumvent some of the complexity of building accurate models for contacts, dynamics, different objects, or external disturbances. Despite their impressive potential for generalizability [23], applications of RL remain limited for real robots due to, for example, sample efficiency [24] and the costs of resetting [25]. Though careful system design has enabled successfully deployed RL systems to learn on locomotion and dexterous manipulation tasks [26, 24], the characteristics of our problem, i.e., fine manipulation with precise contact and hard-to-model dynamics, raise additional challenges in robustness and reactivity.

It is tempting to bypass modeling entirely and directly deploy a model-free RL algorithm for training in the real world. While this approach may eventually learn, it often necessitates tremendous human effort for supervision and resets and is likely to be too inefficient. To make the training more practical, we propose CherryBot, an RL system that combines pre-training in an imprecise simulation with fine-tuning in the real world. With an external state estimation module, CherryBot can be deployed across various scenarios and fulfills the aforementioned requirements:

- **Efficiency:** To enhance sample efficiency for real-world manipulation, we leverage imperfect information readily available to most robots, such as an inaccurate simulator and a heuristic-based baseline policy.
- **Precision and Robustness:** We introduce a challenging task that involves a single *hard-to-grasp* object with *diverse dynamics* for real-world fine-tuning. While this intensifies the difficulty of the task and training, it minimizes the need for human intervention and promotes the learning of robust policies that are resilient to disturbances.
- **Reactiveness:** We carefully design the action space to strike a balance between tractable learning (low-frequency control, slow response, short horizon) and responsiveness (high-frequency control, fast response, long-horizon reasoning). This design optimizes the system's ability to react swiftly while ensuring effective learning.
- **Generalizable**: Our system supports the plug-and-play integration of an external state estimation module even if it introduces certain inaccuracies, allowing deployment across various downstream tasks.

Our work contributes a system that, given only 30 minutes of interaction in the real world, achieves superhuman reactiveness on a dynamic, high-precision task: using chopsticks to grasp a slippery ball swinging in the air. We demonstrate the effectiveness of our system in a variety of evaluation conditions: operating under dynamic disturbances by human or environmental factors, varying perception noise, and changing object shapes and sizes, for which it outperforms a heuristic-based controller that requires hours of tuning. We conduct extensive ablations in a simulator and the real world to provide empirical evidence about how our design choices affect sample efficiency for deploying RL systems in the real world and to verify the robustness of our proposal.

II. Related Works

Dynamic fine grasping. From precondition grasping like DexNet [27] and TransporterNet [28] to a closed-loop, vision-based controller like Qt-Opt [23], most prior works grasped palm-sized objects in a quasi-static table-top setting [29]. In our work, objects are much smaller. Even if the agent intends to grasp across the center of the cherry, for example, the execution alone demands a sub-millimeter precision. Several previous works addressed fine grasping [2, 10] but are limited to static scenarios with rigid surface support. In contrast, we examine a dynamics scene in which failed grasps can potentially move the object. Researchers in the field of dynamic manipulation have developed highly capable systems [30], although these systems come with the drawback of relying on expensive dedicated vision systems. In our approach, however, we prioritize the use of generic and accessible hardware, which is more susceptible to making mistakes. Consequently, it becomes crucial to develop a reactive agent that can effectively recover from such errors. We modify the common evaluation criteria for grasping to allow an agent that fails to get a firm grasp in one shot to continue the trial until it achieves a successful grasp, similar to how humans address dynamic grasping challenge [16].

Reinforcement learning. Despite its wide successes in simulator and locomotion [24, 31, 32], RL has limited real world applications in manipulation. The few successes of applying RL to dexterous manipulation came at the cost of sample inefficiency, which has not made it a preferred choice over model-based control or a hand-designed controller. [23] depends on the large scale of a robot arm farm; [33] requires scalable and dedicated hardware. Previous works

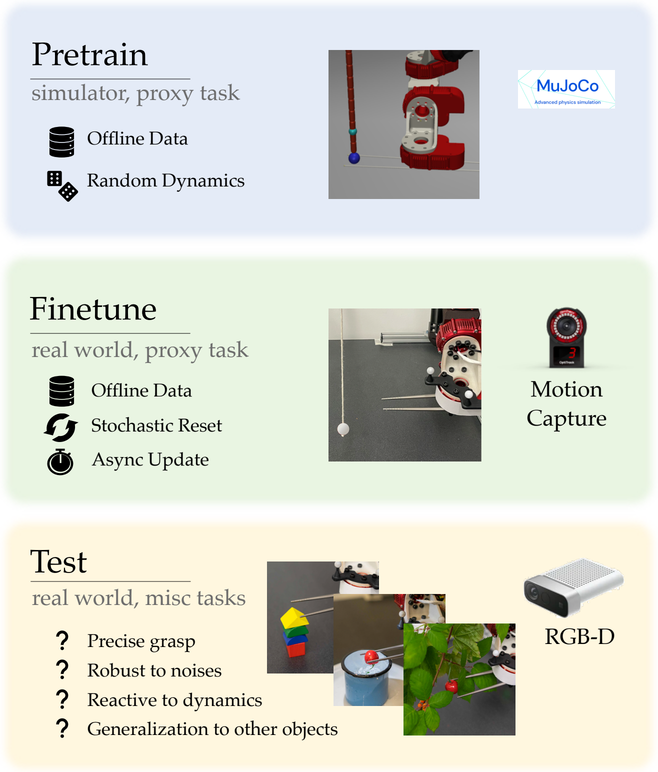


Fig. 3: An overview of the learning framework. The system first pre-trains in simulation and then fine-tunes in the real world. We carefully design the training paradigm to ensure sample efficiency and learning robustness.

focused on model-free RL while our work leverages practical assumptions, including inaccurate simulation [26], imperfect demonstration [34], and normalization techniques [35, 36] to boost sample efficiency for RL and show it is practical on real robots and can achieve superhuman reactiveness.

Offline reinforcement learning (ORL). The recently emerging field of ORL [37, 38] has shown some potential for leveraging offline datasets for RL training [39, 40, 41, 42, 43, 44]. However, whether doing online fine-tuning on top of pre-trained ORL agents could keep improving the performance remains an open question [45].

Visual Servoing (VS). VS controls a robot using real-time visual feedback [46, 47]. It usually requires estimating the pose of the object in the Cartesian space and deriving a control law, such as PID control, directly in the 3D space [48]. Due to the requirement of precision, it is natural to apply VS to fine manipulation tasks. We follow this protocol in the design of our heuristic controller. However, it would require a tremendous amount of expert knowledge and gain-tuning to make the controller generalize across different scenarios or be robust to disturbance and noise. Our baseline controller took hours of gain-tuning but still fails due to disturbances, exemplifying how VS can be sensitive to unexpected dynamics.

III. Method

To address fine manipulation problems in dynamic scenes, we propose a comprehensive framework for efficient training of real-world RL and build a high-performance robotic system. We train a policy to enable a robot equipped with chopsticks to conduct dynamic fine-grasping tasks with non-rigid support. Following established practices in visual servoing [49, 50], we incorporate a separate perception module, using estimated robot and environment states as input to the policy. Notably, our policy includes a *hierarchical controller* (`Hier`) to balance reactivity to dynamics and tractability of learning. At test time, we consider fine grasping of small objects (size $1 \sim 8$ cm) with varying shapes and textures in a dynamic scene: objects might have unstable support and can be moving (e.g., initial velocity, or wind disturbance).

We present a system overview in Fig 3. Initially, we employ pre-training in an approximate simulator with domain randomization (`SimR`), leveraging sub-optimal demonstrations (`Demo`). Subsequently, we fine-tune the system in the real world on a task with a single object (`Proxy`) that presents dynamic challenges but simplifies perception and allows for easy and stochastic reset (`StoRe`). To further enhance training efficiency, we incorporate asynchronous updating (`Async`) and appropriately regularized off-policy RL (`LN`). During testing, the policy integrates an external perception module (`Vis`) to achieve generalization across different objects and scenes.

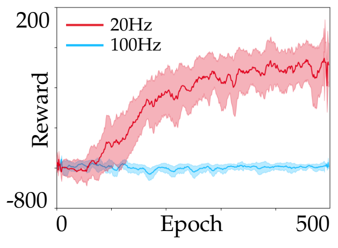
Our contribution lies in instantiating distinct system components to address the combined challenges of real-world RL and dynamic fine manipulation. We outline these challenges in Table I and note how each system component addresses them. To ascertain the impact of each design choice, we conduct ablation studies to measure sample efficiency in both simulation and the real world, where all studies are conducted using 5 consecutive random seeds (Refer to Appendix. VII-B for details). While individual components may have been the subject of previous studies, the combination presented in our work is novel. Our system can grasp a diverse set of small objects in varying dynamic scenarios in the real world, demonstrating the robustness and generalizability to novel objects and disturbances, which could generalize to other tasks with dynamics or requiring fine manipulation.

In this section, we delve into the specifics of our design decisions, grouped into three categories: controller interface design, simulator pre-training and real-world fine-tuning.

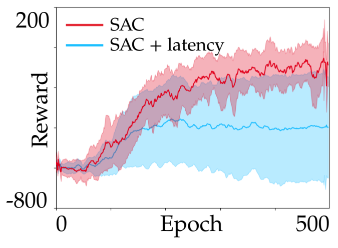
	RL challenges		**Task challenges**			
	Efficient	Reset	Robust	Reactive	Precise	General
`Hier`	✓					
`Demo`	✓					
`SimR`	✓		✓			
`Proxy`	✓	✓	✓	✓	✓	
`StoRe`		✓	✓			
`Async`	✓					
`LN`	✓					
`Vis`	✓					✓

TABLE I: Some challenges inherent to training RL algorithms for dynamic fine manipulation and the design decisions we make to address them.

A. Hierarchical controller of mixed control frequency to balance tractability of learning and reactivity



a Effect of control frequency



b Effect of latency ($< 0.005s$)

Fig. 4: Analysis of using hierarchical mixed-frequency controllers in simulation. **Left:** Impact of different control frequencies on learnability. A hierarchical hybrid-frequency strategy helps balance learnability and policy reactivity. **Right:** Impact of (simulated) latency on performance.

Previous research on dynamic tasks has favored the development of 1000Hz controllers [33, 51] and vision systems [52]. However, as demonstrated in Fig 4a, high-frequency control can negatively impact task horizon and sample efficiency for robotic learning. Real-life RL manipulations commonly employ controllers operating at $5 \sim 15$ Hz for quasi-static tasks [24, 44].

To address this challenge, we adopt a hierarchical controller framework, enabling the RL policy to function at a higher level with a moderate control frequency of 20 Hz, while interpolating commands to a high-frequency (1000Hz) controller. Our system demonstrates that such a controller is capable of solving dynamic tasks and, combined with coherent exploration, makes the learning of a reactive policy tractable.

Furthermore, in the real world, sensor readings suffer from latency and latency can adversely affect reactivity. While [53] demonstrated the advantages of latency randomization in simulation pre-training for a quasi-static task, our own experiments (Fig 4b) revealed that even a small simulated latency (≤ 0.005 s) can harm our task's reactivity to dynamics. Consequently, we made the deliberate decision to exclude simulating latency or latency randomization in simulator training.

Insight: Learning medium-frequency hybrid controllers can effectively balance policy reactivity with the tractability of learning.

B. Leveraging practical information: Approximate simulator `SimR` *and sub-optimal offline data* `Demo`

An RL agent initialized from scratch in the real world makes random movements, frequently encounters safety constraints, and spends a significant portion of training time waiting to be reset. To counter these challenges, we turn to the paradigm of offline pre-training followed by real-world fine-tuning. Where should the data for pre-training actually come from? Since human data collection is expensive for our dynamic fine manipulation task, we instead rely on relatively cheap and abundant, but imperfect, sources of supervision, i.e., approximate simulators and data from heuristic controllers.

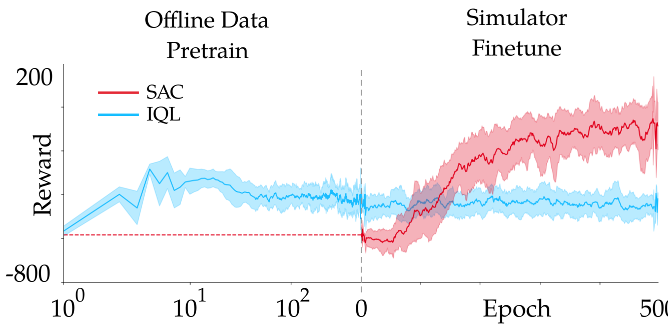


Fig. 5: Analysis of the choice of an off-policy RL algorithm that can learn from offline data while continuing to improve efficiently during fine-tuning.

Choosing an appropriate off-policy algorithm. We first consider an appropriate RL algorithm that can efficiently learn from offline data, simulation samples and online fine-tuning. Although various offline RL methods [42, 43] have been proposed to conduct first pre-training and then fine-tuning, we found that standard *online* RL algorithms (such as soft actor critic (SAC) [32]) are far more effective for fine-tuning than targeted offline RL methods, as Fig 5 shows. In our work, we used variants of soft actor critic [32] for both pre-training and fine-tuning. Our modifications are described below.

Leveraging imperfect data from the simulation. Simulation can provide an appealing source of information since sampling is cheap. However, our task depends heavily on precise contacts and dynamics, but our hardware has varying degrees of kinematic and dynamic errors, making it challenging to construct a precise simulator and making direct simulation-to-reality transfer difficult.

We constructed an approximate simulator, as described in Sec. IV. To best leverage such a simulation that is definitely mismatched with reality, we randomize the dynamics of the physical simulation [53], exposing the agent to a wide distribution of possible physics parameters. Unaware of the changing dynamics of the environment, the agent needs to develop a robust strategy that is conservative to variations in the world dynamics, making it more amenable to real-world fine-tuning. Specifically, we train a Q-function and policy networks with samples from our simulator and then use the networks to initialize the fine-tuning phase. Without pre-training, an RL agent could not improve its task performance even after hours of real-world training, as shown in Fig 6a.

Leveraging sub-optimal offline data. Beyond simulation data, it is useful to provide the agent access to real-world data with sufficient state-action coverage. While it may be expensive to solicit data from a human supervisor or to generate optimal demonstration, it is relatively easy to design a heuristic policy that is sub-optimal but can provide useful state action coverage. In this work, we design a simple heuristic controller that contains a state machine with closed-loop control (see Appendix VII-D). Though this controller has limited reactivity and is not robust to noise, we found that seeding the replay buffer of our RL agent with this sub-optimal data helps with both asymptotic performance and sample efficiency (Fig. 6b).

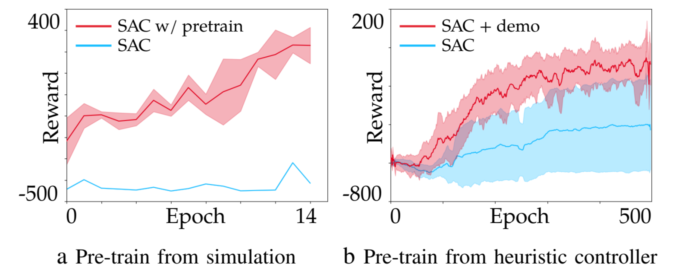


Fig. 6: Analysis of the impact of pre-training for fine manipulation. **(Left)** Having pre-trained in simulation enabled real-world fine-tuning to make progress. **(Right)** Inclusion of offline data collected from a heuristic controller enabled simulator pre-training to succeed on the task in the simulator using many fewer samples. Both techniques significantly aid with learning efficiency.

Insight: Pre-training using standard off-policy RL methods with imperfect prior data from heuristic controllers and simulation can significantly help with sample efficiency for real-world fine-tuning.

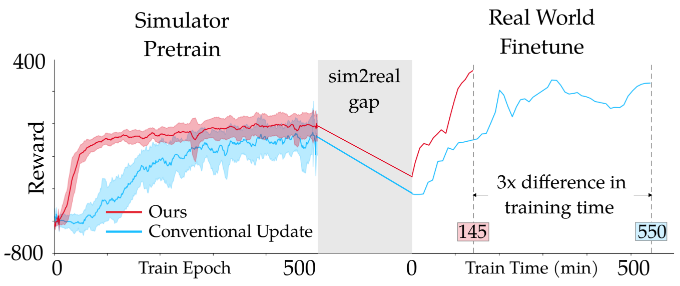


Fig. 7: We measure sample efficiency in simulation by training epochs but in the real world by training time. Combining the asynchronous update, high Update-To-Data (UTD) ratio and LayerNorm regularizer greatly improves the speed of training during the real-world finetuning stage.

C. Fine-tuning in the real world: Proxy task and efficient reinforcement learning

Although pre-training can significantly aid with data efficiency, it is unlikely to achieve optimal performance without further training in the real world. To enable practical fine-tuning in the real world, we focus on two questions: (1) what real-world training setup allows for both ease of training and robust learning, and (2) how do we make this process more efficient?

Pragmatic fine-tuning via a single proxy task. Conventional wisdom might suggest training the RL agent directly on a large variety of test scenarios. However, a real-world training setup would require a prohibitively large number of objects and scenarios to be instrumented, which can be laborious and expensive. Instead, we posit (1) to isolate the perception challenge and control challenge, and (2) first address the control challenge via a `Proxy` task that is appropriately dynamic and difficult, such that it can yield a policy that generalizes to a variety of dynamic scenarios without actually being explicitly exposed to them during training. We identify the criteria needed to set up a proxy task as follows: (1) *representative* of the motion and strategies involved in a broader family of tasks, (2) *reset-friendly*, or having largely autonomous reset, (3) learnable but not trivial, and (4) yields a policy that is *robust.* For our dynamic fine grasping problem, we define the following proxy task: *firmly grip a ball swinging in the air*, as Fig 8b shows. We attach a ball to a thin fishing line to hang it in the air to allow a swinging motion. The task naturally provides a reset since the string resets the ball around its static position due to gravity, obviating the need for a human supervisor to reset the scene.

Exposing the agent to varying initial conditions enables developing a more robust policy: stochastic reset. While waiting for a long period to allow the object to come to rest would yield a *static reset*, we show, surprisingly, that resuming sampling with a randomly moving object enables learning of more dynamic and adaptive behaviors. By embracing this type of *stochastic* reset (`StoR`), we construct a harder-to-learn task, i.e., the agent must react to different initial positions and velocities, so it experiences a larger diversity of conditions during training. As a result, the learned policy becomes more robust to varying dynamic conditions, improving skill transfer to novel disturbances and objects (see Appendix. VII-B for quantitative ablation).

Insight: Designing appropriate proxy tasks to train the agent can simplify the training infrastructure. Additionally, training with stochastic reset can improve the robustness and generalizability of learned policies.

Efficient fine-tuning: Improving gradient throughput with asynchronous updates and regularization

To improve training efficiency in the real world, we care about not only the sample efficiency but also the wall clock time efficiency. In the simulation, operations (e.g., sampling step, resetting hardware) are almost instant. In the real world, however, idle time is unavoidable and adversely affects the speed at which we collect samples: for every second of samples collected, our system spends 4 seconds waiting for a reset. We replace the common training paradigm in simulation, which takes one gradient update after collecting one environment step. We instead perform asynchronous RL updates up to the limit of our computer alongside robot operation. This greatly increases our gradient throughput, effectively committing 10 to 20 updates per data point collected (Update-To-Data, UTD).

Most off-policy RL methods lower their UTD ratios because additional gradient updates lead to unbalanced training of policy/value functions (i.e., exploding actor / value losses). In this work we find that this problem can be significantly mitigated by using appropriate regularization. By using a standard technique such as layer normalization [54], we are able to avoid overfitting and take significantly more gradient steps per data point (See Appendix. VII-B0b). As shown in Fig 7, the combination of asynchronous updates and LayerNorm regularization achieve moderately faster training in simulation but, during fine-tuning on the real robot, significantly boosts the efficiency of learning in terms of wall-clock time.

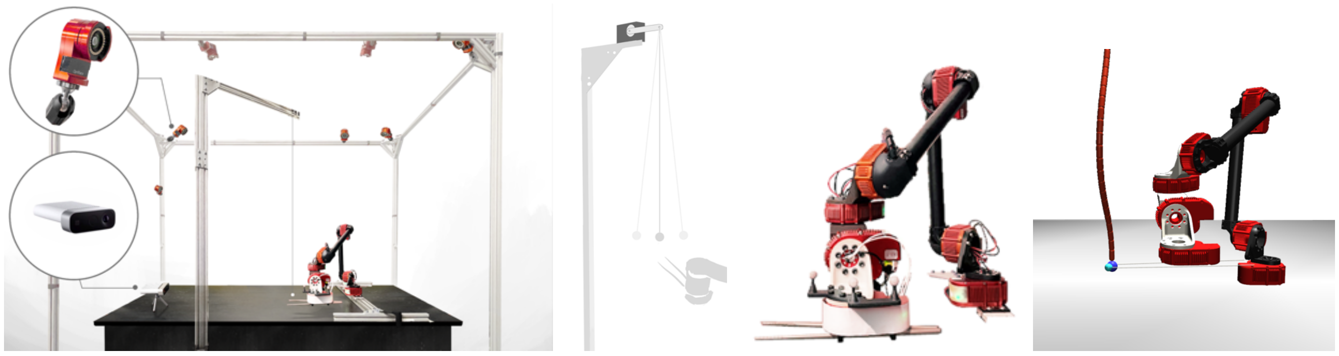

a Hardware overview. b Proxy task. c Assembled robot. d MuJoCo simulation.

Fig. 8: The CherryBot hardware system.

Insights. Improving gradient throughput by leveraging asynchrony and more gradient steps per data point with regularization can make fine-tuning efficient enough for practical real-world use.

IV. Complete System Design for CherryBot

We combine the preceding design decisions and insights into a single system, CherryBot (Fig 3), that can handle challenging dynamic fine manipulation tasks in the real world. Below, we describe the hardware infrastructure and concrete implementation details.

Hardware. Fig. 8a shows an overview of our hardware. We built a 6-DOF robot arm equipped with a pair of chopsticks as its end effector, as shown in Fig. 8c. Since our hardware is assembled from parts with joints that are not strictly rigid, inaccuracies accumulate along robot links. We document our modeling and system identification procedure in Appendix VII-A, where we explain how we used a neural network to predict the residual backlash for each joint, achieving position errors < 3 mm at the robot's end effector.

Reset mechanism. At the end of every sampled trajectory in the real world (average length is about 80 timesteps or 4 seconds), we conduct a fixed trajectory for reset. We (1) slowly raise the robot arm to a set position, (2) touch the string in the proxy task, expecting to constrain its motion and reduce system entropy, and (3) return the robot arm to a fixed start pose and restart the task. The reset process takes about 16 seconds, during which our policy and value networks keep sampling experiences from the replay buffer and conducting RL updates. Notably, the reset leaves the object in a dynamic condition, introducing the dynamism we need for learning robust policies.

Simulation. As described in Section III-B, we leverage simulation for pre-training value functions and policies. We construct the simulation in MuJoCo [55] and perform a system identification procedure. We identify the forward kinematics of the arm and associated dynamics parameters for each joint and built a residual neural network to further improve accuracy, as described in Appendix. VII-A.

Perception. We use different perception modules for training and testing. At training time, we use a motion capture system, Optitrack, to obviate many perception challenges and yield an accurate position for the single object to grasp used in the proxy task (error ~ 0.01mm). At test time, accurate state estimation may not be available. Our system instead accepts any external perception module that can estimate the object's center of mass, following common practice in visual servoing [49, 50]. Noticeably, our system does not necessitate a highly precise estimation module w.r.t. the fine manipulation tasks.

Utilizing an external perception module allows our system to be deployed in various downstream tasks. For illustration, we demonstrate our system using (1) a simple segmentation method with smoothing and (2) an off-shelf detection system, YOLO (both documented in App. VII-B). When the object is static, easy to recognize and the scene is clean, our in-house module detects the object with little error. However, our scenes can have occlusions and can be changing (e.g., cherry shaking in the wind partially occluded by the leaves), and the perception error could reach $\sim$5mm. In our experiment, we will also inject noise to the perception module's output to demonstrate the robustness of our proposal to perception noise.

Additional hardware for dynamic disturbance evaluation. The goal of our fine grasping policies is to firmly grasp the object, despite dynamic disturbances. It is challenging to test this type of robustness without additional machinery to introduce disturbances. To this end, we design a motor disturbance mechanism to systematically inject dynamic disturbance so we could evaluate the robustness of agents. As shown in Fig 10a, the motor perturbs the motion of the ball by rotating the motor's arm and dragging the string attached to it. By varying the string's hanging point, the object can be pulled with different sizes of disturbance (from smaller disturbances to bigger ones, denoted as $20 \sim 100$).

V. Evaluation

In our experiments, we evaluate our proposal on fine manipulation challenges (grasping slippery balls or various small items) and during dynamic disturbances (created using programmed motor movements or human interference). We intend to answer the following questions: (1) Can a policy learned by CherryBot be robust and reactive to dynamic scenarios? (2) Can our policy generalize to different objects?

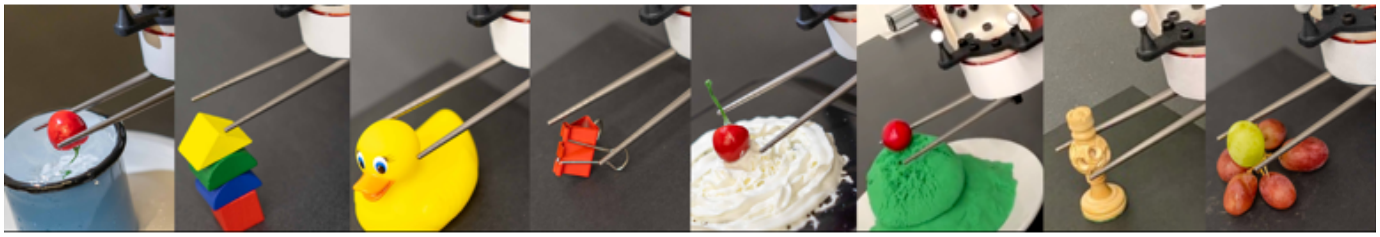
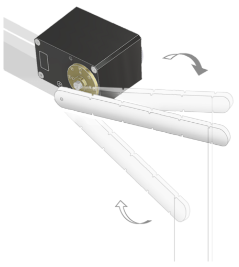


Fig. 9: We deployed our system to generalize to a wide variety of objects with different shapes and textures.

	Marble ball			Cherry			Gaussian noise			Static ball			Dynamic ball		
	Rew.	Succ.	TTS	Rew.	Succ.	TTS	Rew.	Succ.	TTS	Rew.	Succ.	TTS	Rew.	Succ.	TTS
Our RL	**242.3**	**100%**	**1.77**	**321.7**	**100%**	**0.92**	**130.4**	**90%**	**2.06**	**565.3**	**100%**	**0.28**	**512.3**	**100%**	**0.48**
20HZ VS	64.3	60%	1.12	113.1	70%	1.17	-30.9	50%	2.29	196.2	100%	1.49	183.5	90%	1.38
100HZ VS	48.4	50%	0.93	110.7	60%	0.32	109.3	60%	0.64	535.0	100%	0.49	255.0	90%	0.49
Human	-	-	-	-	-	-	-	-	-	-38.8	100%	2.20	-100.2	100%	2.86
Replay	-	-	-	-	-	-	-	-	-	-56.9	80%	1.45	-218.4	50%	2.29
BC	-	-	-	-	-	-	-	-	-	-354.4	30%	0.58	-487.0	10%	0.48

TABLE II: Evaluation results, including average reward (Rew.), success rate (Succ.), and time-to-success for only **successful** trials (TTS in seconds). Note the RL agent is able to adapt to its mistakes and would retry after failure rapidly, therefore achieving a high success rate within the allocated 6 seconds.



a Motor for more dynamic disturbance.

b Human disturbance.

Fig. 10: Dynamic disturbance evaluations we considered. These disturbances test the robustness of the learned CherryBot policies.

(3) Is our proposal robust across random seeds? (4) How do our design decisions impact the final results for CherryBot?

A. Tasks

We test the *reactivity*, *robustness*, and *generalization* of our agent quantitatively and qualitatively. View the video recordings on our website for visualization of our evaluations.

Reactivity to dynamics in the scene. In natural scenarios, a small object might be moved by external disturbance, e.g., a leaf flying in the wind. We simulate this kind of disturbance by installing a motor that pulls the string that suspends the object (Fig. 10a). Additionally, we introduce a more challenging, more spontaneous source of external interference—humans. As shown in Fig. 10b, a person would drag the string while the agent tries to grasp and shake the ball attached to it. We provide a qualitative video on the website reporting the performance of our agent on the latter task.

Robust to perception noise. In the real world, perception errors are inevitable due to, e.g., lightning conditions or occlusions. We simulate such errors by injecting noise into the tracked positions of the object, i.e., adding a small Gaussian noise to the output of our Optitrack system and using this noisy position as the states [56].

Generalization. Different objects have varying shapes, sizes, and textures. We first conduct a quantitative evaluation of our system's generalization ability on grasping hanging cherries with varying shapes and sizes and a hanging marble ball with a slippery surface that made grasping with chopsticks taxing even for humans [16]. Further, we conduct qualitative evaluations on an assortment of objects and dynamic scenarios (Fig. 9), grasping objects of varying shapes (clip, chess), sizes (duck, puzzles) and textures (chess, grapes) and with varying non-rigid support (wind disturbance, floating water, melting cream, falling sand). We further included experiments using an off-the-shelf detection system, YOLO, which empowered our system to grasp objects with more diverse shapes and colors, shown in App. VII-F.

Ablation. During training, we employ a high-accuracy tracking cage that eliminates perception errors, making it an ideal platform for ablation testing. We first place the tracker ball at a fixed position and ask the agent to grasp the **Static** ball, examining how precisely the agent can grasp. Second, we hang the ball and give it a random initial velocity (**Dynamic** ball), testing the agent's reactivity to dynamic scenes.

B. Baselines

We evaluate our agent on five candidate baselines: (1) 20HZ VS, our heuristic-based controller that we used to collect demonstrations, and (2) 100HZ VS: our heuristic controller with a different set of gains. Optionally, we test (3) Humans: a human expert teleoperation the robot, (4) Replay: replaying successful human demonstrations, and (5) Behavior Cloning (BC): a practical imitation learning algorithm [57] trained with our collected data, detailed in App. VII-E.

For each task and each agent we selected, we test over 10 trials and summarize performance statistics. In total, we conducted more than 500 trials. For fair evaluation and to accommodate the slow motion of the human baseline, we use a fixed horizon (6 seconds) at test time.We determine the success of the trajectory by inspecting whether the robot could securely lift the object and stabilize it at a height of 0.1m in the world frame.

C. Reactivity to dynamic disturbance

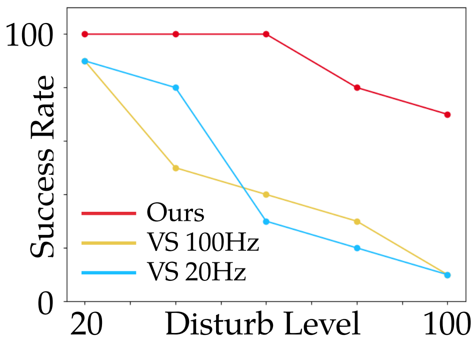


Fig. 11: Success rate under different disturbance. The CherryBot controller's RL agent is robust to disturbances that significantly exceed those for baseline methods.

Fig 11 shows the agents' performance under varying degrees of dynamic disturbance injected by the motor dragging. We observe that our RL agent outperforms the baselines by a large margin across all degrees of motor dragging disturbance. In the human disturbance task, our RL agent succeeded in 17 of 29 trials of human disturbance within the time limit.

D. Robustness to noise

As shown in Table II, when Gaussian noise is applied to the sensor, the RL agent performed the best, achieving a 90% success rate, significantly outperforming the baseline VSs (20Hz achieved 50% and 100Hz achieved 60% success rates). The noisy state estimation is fatal to our vanilla VS baselines which took hours of gain tuning, and it will require further tuning or adding of adaptive modules to improve performance. In contrast, our agent was naturally robust to those noises due to our design choice of `SimR`, `Proxy`, and `StoRe` to train the agent on more challenging tasks before deployment.

E. Generalization

Table II displays our quantitative results on hanging marble balls and cherries, measuring the rewards, success rate and time-to-success on picking up these objects. Our agent significantly outperforms the baselines; the baselines succeeded in some trials quickly, but they could not readjust to their failures to complete the task within the allocated time and instead kept batting the object around.

We further evaluate our agent on more diverse generalization tasks in Fig 9 and show the success rate in Table III. Our

Task	Succ. (%)	Task	Succ. (%)	Task	Succ. (%)
Water Cherry	60 (9/15)	Puzzle	100 (10/10)	Duck	50 (5/10)
Clip	30 (3/10)	Cream Cherry	100 (10/10)	Sand Cherry	53.3 (8/15)
Chess	80 (8/10)	Grapes	70 (7/10)	Tree Cherry	40 (4/10)

TABLE III: Success rates of generalization tasks. CherryBot learned policies and was able to generalize non-trivially across objects.

agent's overall success rate was 64% (of 100 trials), even with our simple perception module. We summarize in four categories the factors explaining the low success rates on some generalization tasks: (1) the object has no firm contact point and keeps slipping out (clip and chess), (2) the object size is too big (duck), (3) the failure of grasping is fatal (cherry drops in the falling-sand task), and (4) perception errors due to occlusion (tree cherry-pick).

F. Further analysis

We examine the performance of all baselines on our proxy task, which removes perception noises but tests the agent's precision and dynamic reactivity. Table II presents results on proxy tasks in both static and dynamic modes. In general, our RL agent outperforms all baselines in success rate, time to succeed, and total rewards. Both the 20Hz and 100Hz VS baselines could also achieve 90 to 100% success rates, respectively, on the proxy tasks, albeit taking a longer time to succeed than our RL agent. It is worth noting that our RL agent, which runs at a hybrid control frequency, uses less time to succeed compared to 100HZ VS. This implies that our hierarchical framework with its low-frequency learned policy develops a more efficient strategy for grasping in the dynamic scene than our hand-designed controller.

Are we cherry-picking a cherry-picking agent? To clarify the reproducibility of our proposal: we use an open-sourced codebase d3rlpy [58], with default implementations and hyperparameters, to run our ablation studies in the simulation and our real-world robotic experiments. We conducted 5 random seed sweeping in simulator ablation studies and 3 random seed sweeping in real-world fine-tuning studies, verifying the the efficacy and reproducibility of our proposal. See Appendix VII-H for details.

VI. Conclusion

We present a system, CherryBot, for learning robust dynamic fine grasping in unstable conditions. We provide empirical evidence demonstrating the effectiveness of our proposed system on a low-cost chopsticks-equipped robot and validate its efficiency, reactivity, robustness and potential to generalize, showing how reinforcement learning can be a practical and competitive tool to learn dynamic and precise behavior.

Despite successfully performing a diverse set of grasping tasks in the real world, there remain significant challenges to address in future work. Expanding our system to a wider range of tasks, including those with longer time horizons, could broaden its applicability. Upgrading the current perception module to include object pose and to have an end-to-end vision-based RL system could enhance its usability and allow further fine-tuning after being deployed. Exploring the theoretical reasons behind the importance of our design decisions, or providing insights into their applicability to different hardware or domains, would also open fascinating areas of future study.

Acknowledgment

We thank Kendal Lowrey and Collin Summers for helping with system identification, Emo Todorov for gifting access to Mujoco, Chris Atkeson, Vikash Kumar for feedback on the draft, Mira Chew and Selest Nashef for 3D design and printing.

This work was (partially) funded by the National Science Foundation NRI (#2132848) and CHS (#2007011), DARPA RACER (#HR0011-21-C-0171), the Office of Naval Research (#N00014-17-1-2617-P00004 and #2022-016-01 UW), and Amazon.

VII. Appendix

A. Hardware

We document how we built a simulation via kinematic modeling and a system identification procedure. We explain how we used a neural network to predict the residual backlash to achieve position errors < 3 mm at the robot's end effector. We also elaborate on our perception module at test time.

a) Modelling and Kinematic Calibration: Performing fine-motor skills requires a carefully calibrated kinematics model that reflects the hardware setup in the real world. The robot manufacturer provides a kinematics model, but this (a) does not account for the chopsticks end-effector, and (b) is not tuned to the degree of accuracy that we require. Therefore, we employ data-driven calibration pipelines that allow us to achieve highly accurate position estimates from joint angles.

The Optitrack cage provides precise and accurate pose information, which we can leverage as ground truth data. When we mount a pair of chopsticks to the robot end-effector, one of them is fixed ("primary" chopsticks) and the other one can be rotated by the end effector joint ("moving" chopsticks). By placing a tracker on the tip of the primary chopstick, we can measure the end-effector position in the 3D space.

First, we need to measure the robot's base position and orientation on the table. We spin the base joint of the robot for several rotations in both directions. This causes the tracker to trace out a circle several times. By measuring the position and inclination of the circle, we can determine the tilt and position of the robot on the table.

We then use teleoperation to control the robot to move inside the workspace, recording the end-effector position and the robot joint angles. We then use this data with either a black-box or gradient-based optimizer (using the factory defaults as the initial guess) to solve for a set of accurate parameters for the kinematics model by minimizing the FK loss. To regularize the optimization, we find that it is important to penalize deviating too far from the manufacturer-given kinematics model. We observed the kinematic error being $1.2 \sim 3$ mm at the end effector tip in the task space we tested.

b) Residual Estimation to Improve Kinematics: To combat inaccuracies in our kinematics, particularly due to backlash in the arm joints, we train a model to predict the backlash in each joint of the arm as a function of the current joint angles. The intuition behind this is that in certain orientations the backlash of the joints may induce errors that are predictable. These dynamics wouldn't be able to be captured by the DH parameters that parameterize the arm kinematics, but a nonlinear function approximator like a neural network would be able to predict this.

The residual estimator network is trained on a dataset of joint angles and grounds truth positions (as determined by the mocap cage) collected in and around the workspace. Additionally, the predicted backlash is constrained to be at most 10% more than the figure listed by the manufacturer, to prevent deviating too far from the kinematic model. In practice, we found that this approach reduced the average position error from 1.8mm to 1.5mm, about a 17% reduction, which is significant given the high-precision nature of the task.

c) System Identification: To build a simulator with dynamics as close to real as possible, we also employ black box optimizer to fit the dynamic transition in our simulator to the real world using recorded trajectories of the robot in and around the workspace. we optimize the environment's parameters (centers of mass of joint bodies, friction coefficients, contact solver parameters, etc.) in order to minimize the divergence of simulated rollouts from the ground truth recording. Again, we regularize this optimization by penalizing the divergence of the simulator parameters from the initial guess.

To run the optimization program efficiently, the code leverages the fast computational speed of Julia and the Lyceum MuJoCo wrapper [59]. We do not expect that this optimization pipeline will result in a simulator that matches reality exactly because of the limitations of the simulator: for example, the shape of the chopsticks and contact parameters are simplified. But it does yield good enough parameters to enable simulator pre-training.

d) Perception Module: Our system can accept any perception module at test time as long as it yields an estimate of the center of the mass of the object to grasp. For demonstration, we built a heuristic-based perception module that estimates the object's centroid position from an Azure Kinect camera. It uses salient pixels to estimate the object's CoM position from RGB-D streams. At test time, the user chooses an object to grasp and specifies a salient color (e.g., red for cherries). We first filter out noise in the image using a Gaussian blur and mask the salient region similar to the desired color. With edge and contour detection, we can get the centroid of the object in pixel space. We then project the 2D point location to 3D using camera intrinsic and calibrated extrinsic matrices.

When the object is static, easy to recognize and the scene is clean, our current perception module detects the object with little error - detecting a red marble on a white background has a 0.3mm error. However, in our experiments, the scenes can have occlusions and can be changing (e.g., cherry shaking in the wind partially occluded by the leaves), and the perception error could reach ≥ 5 mm.

We conduct ablation experiments to verify the robustness of our system to perception noise: After injecting a Gaussian noise with a mean of 5 mm into the perception module, the performance of the visual servo controller dropped from 90% to 50 60%, shown in Table II. Our system. however, achieved 90% even with these noises present, highlighting its robustness to perception noise.

e) Policy Input and Output: Our agent follows [2] and uses the end effector pose (i.e., 3D vector of x-y-z position and 4D quaternion of rotation) plus the perception info (i.e., 3D vector of the center of mass) as input. The high-level policy (20Hz) outputs a command effector pose (3D xyz + 4D rotation), which will be translated by an Inverse Kinematic solver to be a 7D vector of commanded joint positions for the whole arm, including the end effector. Each positional joint command is fed to a lower-level PID controller (1000Hz).

B. System Design Ablations

All ablation experiments mentioned in III were run with the same experimental design to make fair comparisons. Unless otherwise specified in the paper, we focus on the SAC implementation provided by the d3rlpy library, using the default hyperparameters in d3rlpy library, sweeping across the seeds 120, 121, 122, 123, and 124. When run in simulation, we use the simulator whose tuning process is described in Sec. VII-A0c, and real-life experiments are run with the same hardware and perception as described in Sec. IV.

Aside from the ablations studied in depth in Section III, we also performed a few more experiments validating other design choices.

a) Update-to-Data Ratio: The UTD (Update-to-Data ratio) controls the ratio of gradient steps per environment step. Our system achieves an effective UTD=10 through asynchronous updates. We illustrate in Fig. 12 that higher UTD values of 10 and 20 result in much faster convergence than the traditional choice of UTD=1. Running more updates for the same amount of data seems to drastically increase the sample efficiency and expedite learning. The key takeaway is that a large UTD ratio can be favored for its practicality in real-world training with real-time constraints and it showed empirical improvement in learning speed, making real-world learning more tractable.

b) LayerNorm: : LayerNorm is a simple regularization that turns out to be very effective when we employ a high UTD ratio. To show the impact of Layer Norm regularizer, we run trials for it with different UTDs.

c) Stochastic and static reset performance: Our proxy task features a ball swinging from a string which, notably, has a stochastic reset, as described in Section III-C. We illustrate the impact of this design choice via an ablation study in the simulator, training two agents with static or stochastic reset for the same number of epochs. During the evaluation, we test the two agent's performance on the static-reset task and stochastic-reset task. The one trained with stochastic reset yielded an average score of **198.8** on 10 trials, significantly

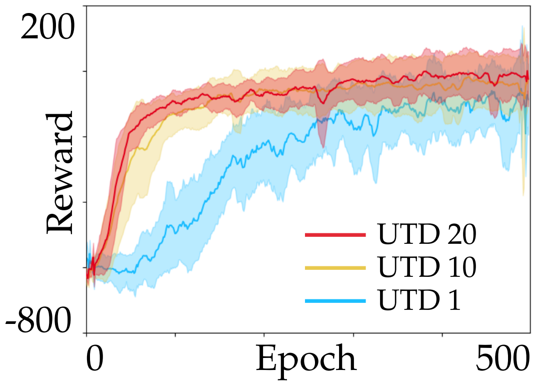


Fig. 12: Training our RL agent in the simulator with varying values of UTD. Higher values of UTD converge much faster than lower ones.

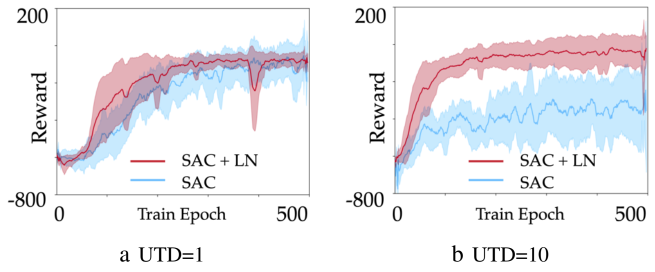


Fig. 13: Analysis of using LayerNorm regularization at different UTD

higher than the agent trained with static reset, which yielded only **127.8**. Evidently, the use of stochastic reset, which yields a larger and more varied initial state distribution, promotes a better-performing policy.

C. Task Design

We use a proxy task (as described in the main paper) as a representative task during our agent's fine-tuning stage and then solve other generalization tasks in zero-shot at deployment time. Instead of "catching" the moving objects directly, we move the agent to be close to the target first and then manipulate it. Thus, the challenging task was divided into two stages: 1) the simple, slow-reaching stage and 2) the reactive grasping stage. During the first stage, assuming the position of the moving objects is known, we use a heuristic policy to approach the object slowly to be within about ~ 3 centimeters away (as the object moves, the distance is not guaranteed to be upper bounded). Then, we will use the learned RL policy or our baseline policies to solve the task in the reactive grasping stage.

Success criteria: The task is considered successful if the robot is able to firmly grasp and lift the correct target object. In the real world, we also used the force torque sensor on the robot joint (which yielded noisy readings) to judge the success of the grasp.

Reward design Prior RL works had reported using both sparse and dense rewards for manipulation tasks to encourage a desired motion [34, 60, 24]. Our reward function also consists of two parts: dense rewards and sparse rewards. The various dense and sparse terms of the reward function are described in Table IV.

Weight	Description
Dense Rewards	
-5	$\|\cdot\|_2$ between object and the chopsticks
-10	$\|\cdot\|_2$ between object and goal
+1	Correctness of distance between chopstick tips
+2	Squeezing torque of chopstick
Sparse Rewards	
-5	Object stayed around initial position
+5	Firmly grasp the object and lift to goal
-10	$\|\cdot\|_2$ between object and chopsticks >10cm

TABLE IV: Rewards design

D. Heuristic controller design

We design our VS(visual servo) controller following human's grasping philosophy in a state-machine manner. Similar to our system, the VS controller also relies on an external state estimation module. First, the robot will align the centroid of the object with the center of chopsticks tips using PD control in 3 axes. It adjusts its pose until the position offset is within a small threshold (1mm), and will then close the chopsticks to attempt the grasp. After that, it will move the object to the goal position with a proportional positional controller. If the robot cannot maintain the height offset within 1 mm during any state, we will return the policy back to the alignment state. The heuristic-based VS controllers are sensitive to gain-tuning and precision of the perception. We spent a reasonable amount of hours tuning the gains of our controller, but there perhaps still remains room for improvement.

E. Imitation Learning

In our early exploration, we experimented with human teleoperators and simple imitation learning. We noted that the replay of the successful demonstration did not have a 100% success rate, suggesting that fine manipulation requires a precise and reactive policy to be robust to potential sensor and actuator errors.

We follow [2] for imitation learning baseline. For the replay, we sample 20 out of 100 successful trajectories generated by human teleoperation. For behavior cloning, we only used the 100 successful trajectories in training. We follow the convention of imitation learning [2, 26] and filter out failed trajectories.

Surprisingly, in both static and dynamic ball setups, BC's performance is even worse than replay, suggesting that the performance of imitation learning is heavily dependent on data support. It is hard to apply BC on these fine manipulation tasks because a tiny divergence between train and test will shift the agent far from a success, motivating us to seed a self-supervised learning method that can learn from trial and error.

F. Off-shelf Perception Module

Our system can plug in any external separate perception module that returns the center-of-mass (CoM) of the object and does not necessitate a high-accuracy perception module. To showcase the robustness of our system, we include experiments using an off-the-shelf detection system, YOLO [61], on our website. YOLO allows our system to grasp objects with more diverse shapes and colors. To use YOLO, we calculate the center of the bounding box returned by YOLO to feed to our system. We ask our system to pick up some plastic figures, exemplifying how to apply our system to handle objects with more complicated shapes and shades.

G. Generalization tasks and their emphasis

Our chosen real-world evaluation tasks effectively capture our agent's ability to generalize from the proxy task to more practical settings. Particularly, these tasks feature real-world complications that are not present during training, allowing us to test the agent's ability to compensate for it at test time.

- Cherry-Picking: Our agent's eponymous cherry-picking task is complicated by noisy perception affected by occlusion. The branches and leaves get in the way of the camera, resulting in biased and noisy estimates of the cherry's position. In succeeding, the agent shows its robustness to overcome these perception challenges.
- Water Cherry, Cream Cherry, Sand Cherry, and Grape: These tasks' main defining traits are the presence of new, unmodeled dynamics. In the case of the cherry floating in the water, if the cherry is disturbed it will float in the direction of the applied force. The cream cherry is relatively less mobile, but the base upon which it sits is deformable. Similarly, the cherry atop the sand is resting on a very unstable surface that is actively falling apart over time. Finally, the grape is on top of a pile of other grapes which tends to collapse when disturbed. In all four cases, there are new dynamics that are completely unrepresentative of conditions at train time. Evidently, the agent is able to generalize to these new dynamics and successfully solve the task.
- Clip, Chess, Puzzle, and Duck: These tasks showcase the agent's ability to grasp objects with completely different shapes. The agent was trained only to grasp small spheres, but in succeeding in these tasks, it shows that the agent still learns a grasping strategy that is widely applicable to other types of shapes.

H. Robustness to random seeds and hyperparameters

We used a standard, open-source implementation of RL algorithms, d3rlpy, to run our ablation studies in the simulation and our real-world robotic experiments. We use the default implementation of SAC and IQL in d3rlpy alongside the default hyperparameters. The only edits we make are to enable high UTD and LayerNorm. We did not change the hyperparameters or the algorithms when we ran the experiment. To enable d3rlpy to run on the real robot: we changed the location of the action normalization function in d3rlpy to ensure it will be called both for the offline dataset and online finetuning on a real robot.

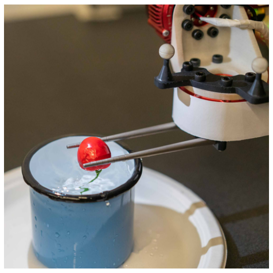
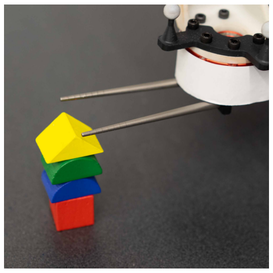
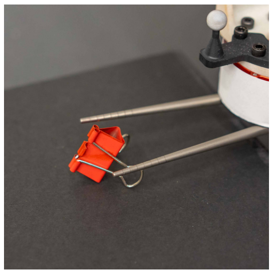
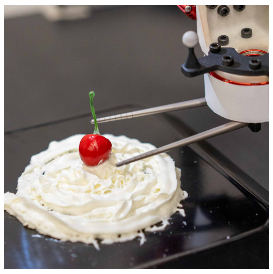
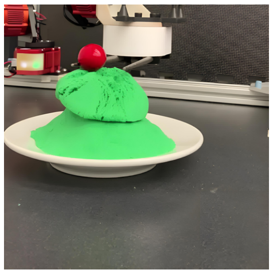
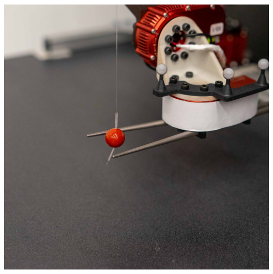
Figure	Task	Succ. (%)	Figure	Task	Succ. (%)
	Water Cherry	60 (9/15)		Puzzle	100 (10/10)
	Duck	50 (5/10)		Clip	30 (3/10)
	Cream Cherry	100 (10/10)		Sand Cherry	53.3 (8/15)
	Chess	80 (8/10)		Grapes	70 (7/10)
	Tree Cherry	40 (4/10)		Marble Ball	100

TABLE V: Visualization and Success rates of testing tasks.

Prior to running the experiments we fixed the random seed. To verify the significance of our findings in the ablation study, we conducted seed sweeping in the simulator, with 5 consecutive random seeds $120 \sim 124$, as shown in 7. To verify the reproducibility of our proposed system in the real world, we trained our whole system including pretraining and fine-tuning with seed $121 \sim 124$), the performance shown in 6a is summarized over all random seeds ran in the real world.

References

[1] Mark R Cutkosky. *Robotic grasping and fine manipulation*, volume 6. Springer Science & Business Media, 2012.

[2] Liyiming Ke, Jingqiang Wang, Tapomayukh Bhattacharjee, Byron Boots, and Siddhartha Srinivasa. Grasping with chopsticks: Combating covariate shift in model-free imitation learning for fine manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6185–6191. IEEE, 2021.

[3] Matthew T Mason and Kevin M Lynch. Dynamic manipulation. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 1, pages 152–159. IEEE, 1993.

[4] Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446), 2019.

[5] Col Michael R Marohn and Capt Eric J Hanly. Twenty-first century surgery using twenty-first century technology: Surgical robotics. *Current Surgery*, 61(5):466–473, 2004.

[6] Wenzhen Yuan, Siyuan Dong, and Edward H Adelson. Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors*, 17(12):2762, 2017.

[7] Tapomayukh Bhattacharjee, Gilwoo Lee, Hanjun Song, and Siddhartha S Srinivasa. Towards robotic feeding: Role of haptics in fork-based food manipulation. *IEEE Robotics and Automation Letters*, 4(2):1485–1492, 2019.

[8] Shuguang Li, John J Stampfli, Helen J Xu, Elian Malkin, Evelin Villegas Diaz, Daniela Rus, and Robert J Wood. A vacuum-driven origami "magic-ball" soft gripper. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7401–7408. IEEE, 2019.

[9] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *The International Journal of Robotics Research*, 41(7):690–705, 2022.

[10] Minho Hwang, Jeffrey Ichnowski, Brijen Thananjeyan, Daniel Seita, Samuel Paradis, Danyal Fer, Thomas Low, and Ken Goldberg. Automating surgical peg transfer: Calibration with deep learning can exceed speed, accuracy, and consistency of humans. *IEEE Transactions on Automation Science and Engineering*, 2022.

[11] Kevin M Lynch and Matthew T Mason. Dynamic nonprehensile manipulation: Controllability, planning, and experiments. *The International Journal of Robotics Research*, 18(1):64–92, 1999.

[12] Haruka Sakurai, Takahiro Kanno, and Kenji Kawashima. Thin-diameter chopsticks robot for laparoscopic surgery. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4122–4127, 2016.

[13] Rohan A Joseph, Alvin C Goh, Sebastian P Cuevas, Michael A Donovan, Matthew G Kauffman, Nilson A Salas, Brian Miles, Barbara L Bass, and Brian J Dunkin. Chopstick surgery: a novel technique improves surgeon performance and eliminates arm collision in robotic single-incision laparoscopic surgery. *Surgical Endoscopy*, 24(6): 1331–1335, 2010.

[14] Akira Yamazaki and Ryosuke Masuda. Autonomous foods handling by chopsticks for meal assistant robot. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6. VDE, 2012.

[15] Ahmed A Ramadan, Tomohito Takubo, Yasushi Mae, Kenichi Oohara, and Tatsuo Arai. Developmental process of a chopstick-like hybrid-structure two-fingered micromanipulator hand for 3-d manipulation of microscopic objects. *IEEE Transactions on Industrial Electronics*, 56 (4):1121–1135, 2009.

[16] Liyiming Ke, Ajinkya Kamat, Jingqiang Wang, Tapomayukh Bhattacharjee, Christoforos Mavrogiannis, and Siddhartha S Srinivasa. Telemanipulation with chopsticks: Analyzing human factors in user demonstrations. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11539–11546. IEEE, 2020.

[17] Matthew T Mason, Siddhartha S Srinivasa, and Andres S Vazquez. Generality and simple hands. In *Robotics Research*, pages 345–361. Springer, 2011.

[18] Bao-Chi Chang, Biing-Shiun Huang, Ching-Kong Chen, and Shyh-Jen Wang. The pincer chopsticks: The investigation of a new utensil in pinching function. *Applied ergonomics*, 38(3):385–390, 2007.

[19] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (ToG)*, 31 (4):1–8, 2012.

[20] Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE, 2016.

[21] Francois R Hogan and Alberto Rodriguez. Reactive planar non-prehensile manipulation with hybrid model predictive control. *The International Journal of Robotics Research*, 39(7):755–773, 2020.

[22] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. *Adaptive motion of animals and machines*, pages 261–280, 2006.

[23] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[24] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657. IEEE, 2019.

[25] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real world robotic reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL

https://openreview.net/forum?id=rJe2syrtvS.

[26] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

[27] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964. IEEE, 2016.

[28] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR, 2021.

[29] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Yale-CMU-Berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017.

[30] Koichi Hashimoto, Akio Namiki, and Masatoshi Ishikawa. A visuomotor control architecture for high-speed grasping. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228)*, volume 1, pages 15–20. IEEE, 2001.

[31] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[32] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[33] Manuel Wüthrich, Felix Widmaier, Felix Grimminger, Joel Akpo, Shruti Joshi, Vaibhav Agrawal, Bilal Hammoud, Majid Khadiv, Miroslav Bogdanovic, Vincent Berenz, et al. Trifinger: An open-source robot for learning dexterity. *arXiv preprint arXiv:2008.03596*, 2020.

[34] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[35] Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. *arXiv preprint arXiv:2101.05982*, 2021.

[36] Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022.

[37] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. *Reinforcement learning: State-of-the-art*, pages 45–73, 2012.

[38] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[39] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[40] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. corr abs/1812.02900 (2018). *arXiv preprint arXiv:1812.02900*, 2018.

[41] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

[42] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[43] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[44] Gaoyue Zhou, Liyiming Ke, Siddhartha Srinivasa, Abhinav Gupta, Aravind Rajeswaran, and Vikash Kumar. Real world offline reinforcement learning with realistic data source. *arXiv preprint arXiv:2210.06479*, 2022.

[45] Joey Hong, Aviral Kumar, and Sergey Levine. Confidence-conditioned value functions for offline reinforcement learning. *arXiv preprint arXiv:2212.04607*, 2022.

[46] François Chaumette and Seth Hutchinson. Visual servo control. I. Basic approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, 2006.

[47] Seth Hutchinson, Gregory D Hager, and Peter I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996.

[48] Arthur C Sanderson and Lee E Weiss. Adaptive visual servo control of robots. *Robot vision*, pages 107–116, 1983.

[49] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[50] Xiaohan Zhang, Yifeng Zhu, Yan Ding, Yuke Zhu, Peter Stone, and Shiqi Zhang. Visually grounded task and motion planning for mobile manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1925–1931. IEEE, 2022.

[51] Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Iterative residual policy: for goal-conditioned dynamic manipulation of deformable objects. *arXiv preprint arXiv:2203.00663*, 2022.

[52] Kohei Okumura, Hiromasa Oku, and Masatoshi Ishikawa.

High-speed gaze controller for millisecond-order pan/tilt camera. In *2011 IEEE International Conference on Robotics and Automation*, pages 6186–6191. IEEE, 2011.

[53] 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)*, pages 3803–3810. IEEE, 2018.

[54] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[55] 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, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.

[56] Russ Tedrake. *Underactuated Robotics*. 2023. URL https://underactuated.csail.mit.edu.

[57] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[58] Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315):1–20, 2022. URL http://jmlr.org/papers/v23/22-0017.html.

[59] Colin Summers, Kendall Lowrey, Aravind Rajeswaran, Siddhartha Srinivasa, and Emanuel Todorov. Lyceum: An efficient and scalable ecosystem for robot learning. In *Learning for Dynamics and Control*, pages 793–803. PMLR, 2020.

[60] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pages 4344–4353. PMLR, 2018.

[61] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.