Efficient Bimanual Manipulation Using Learned Task Schemas

Rohan Chitnis* **Shubham Tulsiani** **Saurabh Gupta** **Abhinav Gupta**

MIT Computer Science and Artificial Intelligence Laboratory, Facebook Artificial Intelligence Research
ronuchit@mit.edu, shubhtuls@fb.com, saurabhg@illinois.edu, gabhinav@fb.com

***Abstract*— We address the problem of effectively composing skills to solve sparse-reward tasks in the real world. Given a set of parameterized skills (such as exerting a force or doing a top grasp at a location), our goal is to learn policies that invoke these skills to efficiently solve such tasks. Our insight is that for many tasks, the learning process can be decomposed into learning a state-independent task schema (a sequence of skills to execute) and a policy to choose the parameterizations of the skills in a state-dependent manner. For such tasks, we show that explicitly modeling the schema's state-independence can yield significant improvements in sample efficiency for model-free reinforcement learning algorithms. Furthermore, these schemas can be transferred to solve related tasks, by simply re-learning the parameterizations with which the skills are invoked. We find that doing so enables learning to solve sparse-reward tasks on real-world robotic systems very efficiently. We validate our approach experimentally over a suite of robotic bimanual manipulation tasks, both in simulation and on real hardware. See videos at `http://tinyurl.com/chitnis-schema`.**

I. Introduction

Let us consider the task of opening a bottle. How should a two-armed robot accomplish this? Even without knowing the bottle geometry, its position, or its orientation, one can answer that the task will involve holding the bottle's base with one hand, grasping the bottle's cap with the other hand, and twisting the cap off. This "schema," the high-level plan of what steps need to be executed, only depends on the task and not on the object's geometric and spatial state, which only influence how to parameterize each of these steps (e.g., deciding where to grasp, or how much to twist).

Reinforcement learning methods provide a promising approach for learning such behaviors from raw sensory input [1], [2], [3]. However, typical end-to-end reinforcement learning methods do not leverage the schematics of tasks, and instead aim to solve tasks by learning a policy, which would involve inferring both the schema and the parameterizations, as a function of the raw sensory input. These approaches have led to impressive successes across domains such as game-playing [1], [4], [5], [6] and robotic control tasks [2], [7], [8], [9], but are known to have very high sample complexity. For instance, they require millions of frames of interaction to learn to play Atari games, or several weeks' worth of experience to learn simulated control policies, which makes them impractical to train on real hardware.

In this work, we address the problem of learning to perform tasks in environments with a sparse reward signal, given a discrete set of generic *skills* parameterized by continuous arguments. Examples of skills include exerting a force at a location or moving an end effector to a target pose. Thus, the action space is hybrid discrete-continuous [10]: at each timestep, the agent must decide both 1) which skill to use and 2) what continuous arguments to use (e.g., the location to apply force, the amount of force, or the target pose to move to). The sample inefficiency of current reinforcement learning methods is exacerbated in domains with these large search spaces; even basic tasks such as opening a bottle with two arms are challenging to learn from sparse rewards. While one could hand-engineer dense rewards, this is undesirable as it does not scale to more complicated tasks. We ask a fundamental question: can we use the given skills to efficiently learn policies for tasks with a large policy search space, like bimanual manipulation, given only sparse rewards?

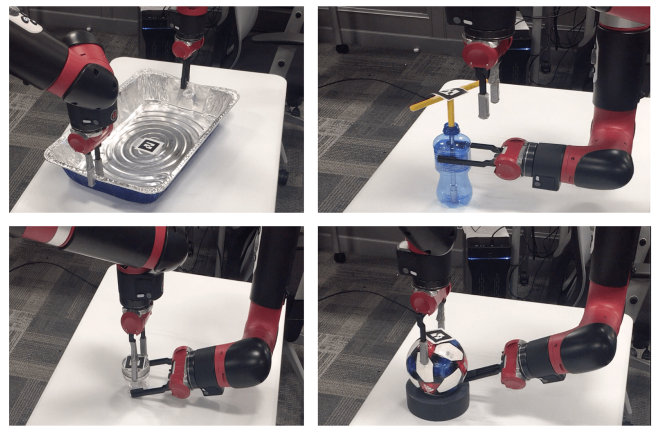

Fig. 1: We learn policies for solving real-world sparse-reward bimanual manipulation tasks from raw RGB image observations. We decompose the problem into learning a state-independent task schema (a sequence of skills to execute) and a state-dependent policy that appropriately instantiates the skills in the context of the environment. This decomposition speeds up learning and enables transferring schemas from simulation to the real world, leading to successful task execution within a few hours of real-world training. The AR tags are only used for automating the reward computation; our model is not given any object pose information. *Top left*: Lifting an aluminum tray. *Top right*: Rotating a T-wrench. *Bottom left*: Opening a glass jar. *Bottom right*: Picking up a large soccer ball.

Our insight is that for many tasks, the learning process can be decomposed into learning a *state-independent* task schema (sequence of skills) and a *state-dependent* policy that chooses appropriate parameterizations for the different skills. Such a decomposition of the policy into state-dependent

* Work done during an internship at Facebook AI Research.

and state-independent parts simplifies the credit assignment problem and leads to more effective sharing of experience, as data from different instantiations of the task can be used to improve the same shared skills. This leads to faster learning.

This modularization can further allow us to *transfer* learned schemas among related tasks, *even if they have different state spaces*. For example, suppose we have learned a good schema for picking up a long bar in simulation, where we have access to object poses, geometry information, etc. We can then reuse that schema for a related task such as picking up a tray in the real world from only raw camera observations, even though both the state space and the optimal parameterizations (e.g., grasp poses) differ significantly. As the schema is fixed, policy learning for this tray pickup task will be very efficient, since it only requires learning the (observation-dependent) arguments for each skill. Transferring the schema in this way enables learning to solve sparse-reward tasks very efficiently, making it feasible to train real robots to perform complex skills. See Figure 2 for an overview of our approach.

We validate our approach over a suite of robotic bimanual manipulation tasks, both in simulation and on real hardware. We give the robots a very generic library of skills such as twisting, lifting, and reaching. Even given these skills, bimanual manipulation is challenging due to the large search space for policy optimization. We consider four task families: lateral lifting, picking, opening, and rotating, all with varying objects, geometries, and initial poses. All tasks have a sparse binary reward signal: 1 if the task is completed, and 0 otherwise. We empirically show that a) explicitly modeling schema state-independence yields large improvements in learning efficiency over the typical strategy of conditioning the policy on the full state, and b) transferring learned schemas to real-world tasks allows complex manipulation skills to be discovered within only a few hours ($<$10) of training on a single setup. Figure 1 shows some examples of real-world tasks solved by our system.

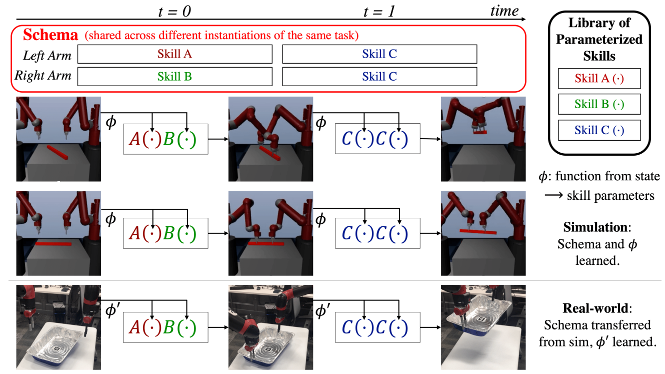

Fig. 2: In bimanual manipulation tasks, a schema is a sequence of skills for each arm to execute. We train in simulation both a state-independent model for predicting this schema and a state-dependent neural network ϕ for predicting its continuous parameters, via reinforcement learning. Our experiments show that using a state-independent schema predictor for these tasks makes training significantly more efficient. To solve real-world tasks, we transfer schemas learned in simulation, and only optimize ϕ.

II. Related Work

Search in parameterized action spaces. An agent equipped with a set of skills parameterized by continuous arguments must learn a policy that decides both which skills to use and what continuous arguments to use for them. Parameterized action MDPs (PAMDPs) [10] were constructed for this exact problem setting. Recent work has addressed deep reinforcement learning for PAMDPs [11], [12], by learning policies that output both the discrete skill and continuous parameter selections at each timestep. In contrast, we propose a model that bakes in state-independence of the discrete skill selection, and show that this assumption not only improves learning efficiency, but also is experimentally useful. A separate line of work learns control policies for steps in a *policy sketch* [13], which can be recombined in novel ways to solve new task instances; however, this work does not consider the discrete search aspect of the problem, as we do.

Transfer learning for robotics. The idea of transferring a learned policy from simulation to the real world for more efficient robotic learning was first developed in the early 1990s [14], [15]. More recent techniques include learning from model ensembles [16] and utilizing domain randomization [17], [18], [19], in which physical properties of a simulated environment are randomized to allow learned policies to be robust. However, as these methods directly transfer the policy learned in simulation, they rely on the simulation being visually and physically similar to the real world. In contrast, we only transfer one part of our learned policy — the skill sequence to be executed — from simulation to the real world, and allow the associated continuous parameters to be learned in the real-world domain.

Temporal abstraction for reinforcement learning. The idea of using temporally extended actions to reduce the sample complexity of reinforcement learning algorithms has been studied for decades [20], [21], [22], [23]. For instance, work on macro-actions for MDPs [23] attempts to build a hierarchical model in which the primitive actions occupy the lowest level, and subsequently higher levels build local policies, each equipped with their own termination conditions, that make use of actions at the level below. More recent work seeks to learn these hierarchies [24], [25], but successes have largely been limited to simulated domains due to the large amount of data required. In our work, we propose a model that makes skill selection independent of state, enabling real-world robotic tasks to be solved via transfer.

III. Approach

Given a set of parameterized skills, we aim to solve sparse-reward tasks by learning a policy that decides both which skill to execute and what arguments to use when invoking it. Our insight is that, for many tasks, the same sequence of skills (possibly with different arguments) can be used to optimally solve different instantiations of the task. We operationalize this by disentangling the policy into a state-independent task schema (sequence of skills) and a state-dependent prediction of how to parameterize these skills. We first formally define our problem setup, and then present our model for leveraging the state-independence of schemas to learn efficiently. Finally, we describe how our approach also allows transferring schemas across tasks, letting us learn real-world policies from raw images by reusing schemas learned for related tasks in simulation.

Problem Setup. Each task we consider is defined as a parameterized action Markov decision process (PAMDP) [10], [26] with finite horizon T. The reward for each task is a binary function indicating whether the current state is an element of the set of desired goal configurations, such as a state with the bottle opened. The learning objective, therefore, is to obtain a policy π that maximizes the expected proportion of times that following it achieves the goal. Note that this is a particularly challenging setup for reinforcement learning algorithms due to the sparsity of the reward function.

The agent is given a discrete library of generic *skills* $\mathcal{X}$, where each skill $x \in \mathcal{X}$ is parameterized by a corresponding vector v^x of continuous values. Examples of skills can include exerting a force at a location, moving an end effector to a target pose, or rotating an end effector about an axis. Let $\mathcal{A}$ denote the action space of the PAMDP. An action $a \in \mathcal{A}$ is a tuple $\langle x, v^x \rangle$, indicating what skill to apply as well as the corresponding parameterization. A *schema* $\bar{x}$ is a sequence of T skills in $\mathcal{X}$, where $\bar{x} = x_1, x_2, ..., x_T$ captures the sequence of skills but not their continuous parameters.

Assumption. We assume that the optimal schema $\bar{x}^*$ is state-independent: it depends only on the task, not on the state and its dynamics. This implies that the same schema is optimal for all instantiations of a task, e.g. different geometries and poses of objects. We note that this is a valid assumption across many tasks of interest, since the skills themselves can be appropriately chosen to be complicated and expressive, such as stochastic, closed-loop control policies for guiding an end effector.

Modular Policies. The agent must learn a policy π that, at each timestep, infers both which skill $x \in \mathcal{X}$ to use (a discrete choice) and what continuous arguments v^x to use.

What is a good form for such a policy? A simple strategy, which we use as a baseline and depict in Figure 3 (top), would be to represent π via a neural network, with weights ϕ, that takes the state as input and has a two-headed output. One head predicts logits that represent a categorical distribution over the skills $\mathcal{X}$, while the other head predicts a mean and variance of a Gaussian distribution over continuous argument values for all skills. To sample an action, we can sample $x \in \mathcal{X}$ from the logits predicted by the first head, then sample arguments using the subset of means and variances predicted by the second head that correspond to v^x.

However, this does not model the fact that the optimal schema is state-independent. To capture this, we need to remove the dependence of the discrete skill selection on the input state. Thus, we propose to maintain a separate $T \times |\mathcal{X}|$ array, where row t is the logits of a categorical distribution over which skill to use at time t. Note that T is the horizon of the MDP. In this architecture, the neural network is only tasked with predicting the skill arguments. The $T \times |\mathcal{X}|$ array of logits and the neural network, taken together, represent the policy π, as depicted in Figure 3 (bottom).

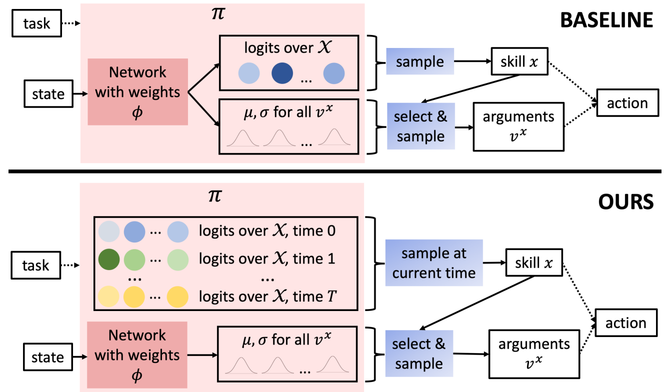

Fig. 3: *Top:* A baseline policy architecture for solving tasks with a discrete set of skills $\mathcal{X}$, each parameterized by a vector v^x of continuous values. The policy for a task is a neural network with weights ϕ that predicts 1) logits over which skill to use and 2) mean and variance of a Gaussian over continuous argument values for all skills. To sample an action, we first sample a skill based on the logits, then select the corresponding subset of continuous arguments, and finally sample argument values from those Gaussians. *Bottom:* Our proposed policy architecture, which leverages the assumption that the optimal schema is state-independent. The key difference is that the neural network only predicts a distribution over continuous argument values, and we train a state-independent $T \times |\mathcal{X}|$ array of logits over which skill to use at each timestep. In the text, we discuss how to update these logits.

Learning Schemas and Skill Arguments. The weights ϕ of the neural network can be updated via standard policy gradient methods. Let τ denote a trajectory induced by following π in an episode. The objective we wish to maximize is $J(\phi) \equiv \mathbb{E}_\tau[r(\tau)]$. Policy gradient methods such as REINFORCE [27] leverage the likelihood ratio trick, which says that $\nabla_\phi J(\phi) = \mathbb{E}_\tau[r(\tau)\nabla_\phi \log \pi(\tau)]$, to tune ϕ via gradient ascent. When estimating this gradient, we treat the current setting of the array of logits as a constant.

Updating the logits within the $T \times |\mathcal{X}|$ array can also be achieved via policy gradients; however, since there is no input, and because we have sparse rewards, the policy optimization procedure is quite simple. Let φ_{tx} be the logit for time t and skill x. Given trajectory $\tau = \langle s_0, x_0, v_0^{x_0}, s_1, x_1, v_1^{x_1}, ..., s_T \rangle$:

- If τ achieves the goal, i.e. $r(\tau) > 0$, increase φ_{tx} for each timestep t and skill x taken at that timestep.
- If τ does not achieve the goal, i.e. $r(\tau) = 0$, decrease φ_{tx} for each timestep t and skill x taken at that timestep.

The amount by which to increase or decrease φ_{tx} is absorbed by the step size and thus gets tuned as a hyper-parameter. See Algorithm 1 for full pseudocode.

Schema Transfer Across Tasks. Since we have disentangled the learning of the schema from the learning of the skill arguments within our policy architecture, we can now *transfer* the $T \times |\mathcal{X}|$ array of logits across related tasks, as long as the skill spaces and horizons are equal. Therefore, learning for a new task can be made efficient by reusing a previously

Task Family	Object (Sim)	Objects (Real)	Schema Discovered from Learning in Simulation
lateral lifting	bar	aluminum tray, rolling pin, heavy bar, plastic box	**1)** L: top grasp, R: top grasp **2)** L: lift, R: lift
picking	ball	soccer ball	**1)** L: top grasp, R: go-to pose **2)** L: no-op, R: go-to pose **3)** L: lift, R: lift
opening	bottle	glass jar, water bottle	**1)** L: top grasp, R: side grasp **2)** L: twist, R: no-op
rotating	corkscrew	T-wrench, corkscrew	**1)** L: go-to pose, R: side grasp **2)** L: go-to pose, R: no-op **3)** L: rotate, R: no-op

TABLE I: Task families, object considered in simulation, objects considered in real world, and schemas (for left and right arms) discovered by our algorithm in simulation. Schemas learned in simulation for a task family are transferred to multiple objects in the real world.

Algorithm TRAIN-POLICY($\mathcal{M}, \alpha, \beta$)

Input: $\mathcal{M}$, an MDP as defined in Section III.
Input: α and β, step sizes.
Initialize neural network weights ϕ.
Zero-initialize $T \times |\mathcal{X}|$ array of logits φ_{tx}.
while *not done* **do**
  $\mathcal{D} \leftarrow$ batch of trajectories $\tau = \langle s_t, x_t, v_t^{x_t} \rangle$ obtained from running policy π in $\mathcal{M}$.
  $\nabla_\phi J(\phi) \leftarrow$ POLICYGRADIENT($\pi, \mathcal{D}$)
  $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$ `// Or Adam [28].`
  for *each trajectory* $\tau \in \mathcal{D}$ **do**
    for *each skill* x_t *used in* τ **do**
      if τ *achieves the goal* **then**
        $\varphi_{tx} \leftarrow \varphi_{tx} + \alpha$
      else
        $\varphi_{tx} \leftarrow \varphi_{tx} - \beta$

Algorithm 1: Training policies π that explicitly model the state-independence of schemas via a $T \times |\mathcal{X}|$ array of logits over what skill to use at each timestep.

learned schema, since we would only need to train the neural network weights ϕ to infer skill arguments for that new task.

Importantly, transferring the schema is reasonable even when the tasks have *different state spaces*. For instance, one task can be a set of simulated bimanual bottle-opening problems in a low-dimensional state space, while the other involves learning to open bottles in the real world from high-dimensional camera observations. As the state spaces can be different, it follows immediately that the tasks can also have different optimal arguments for the skills.

IV. EXPERIMENTS

We test our proposed approach on four robotic bimanual manipulation task families: lateral lifting, picking, opening, and rotating. Table I lists the different objects that we considered for each one. These task families were chosen because they represent a challenging hybrid discrete-continuous search space for policy optimization, while meeting our requirement that the optimal schema is independent of the state. We show results on these tasks both in simulation and on real Sawyer arms: schemas are learned in simulation by training with low-dimensional state inputs, then transferred as-is to visual inputs (in simulation as well as in the real world), for which we only need to learn skill arguments. Our experiments show that our proposed approach is significantly more sample-efficient than one that uses the baseline policy architecture, and allows us to learn bimanual policies on real robots in less than 10 hours of training. We first describe the experimental setup, then discuss our results.

A. MuJoCo Experimental Setup

Environment. For all four task families, two Sawyer robot arms with parallel-jaw grippers are placed at opposing ends of a table, facing each other. A single object is placed on the table, and the goal is to manipulate the object's pose in a task-specific way. *Lateral lifting (bar):* The goal is to lift a heavy and long bar by 25cm while maintaining its orientation. We vary the bar's location and density. *Picking (ball):* The goal is to lift a slippery (low coefficient of friction) ball vertically by 25cm. The ball slips out of the gripper when grasped by a single arm. We vary the ball's location and coefficient of friction. *Opening (bottle):* The goal is to open a bottle implemented as two links (a base and a cap) connected by a hinge joint. If the cap is twisted without the base being held in place, the entire bottle twists. The cap must undergo a quarter-rotation while the base maintains its pose. We vary the bottle's location and size. *Rotating (corkscrew):* The goal is to rotate a corkscrew implemented as two links (a base and a handle) connected by a hinge joint, like the bottle. The handle must undergo a half-rotation while the base maintains its pose. We vary the corkscrew's location and size.

Skills. The skills we use are detailed in Table II, and the search spaces for the skill parameters are detailed in Table III. Note that because we have two arms, we actually need to search over a cross product of this space with itself.

State and Policy Representation. Experiments conducted in the MuJoCo simulator [29] use a low-dimensional state: proprioceptive features (joint positions, joint velocities, end effector pose) for each arm, the current timestep, geometry information for the object, and the object pose in the world frame and each end effector's frame. The policy is represented as a 4-layer MLP with 64 neurons in each layer, ReLU activations, and a multi-headed output for the actor and the critic. Since object geometry and pose can only be computed within the simulator, our real-world experiments will instead use raw RGB camera images.

Training Details. We use the Stable Baselines [30] implementation of proximal policy optimization (PPO) [31], though our method is agnostic to the choice of policy gradient algorithm. We use the following hyper-parameters: Adam [28] with learning rate 0.001, clipping parameter 0.2, entropy loss coefficient 0.01, value function loss coefficient

Skill	Allowed Task Families	Continuous Parameters
top grasp	lateral lifting, picking, opening	(x, y) position, z-orientation
side grasp	opening, rotating	(x, y) position, approach angle
go-to pose	picking, rotating	(x, y) position, orientation
lift	lateral lifting, picking	distance to lift
twist	opening	none
rotate	rotating	rotation axis, rotation radius
no-op	all	none

TABLE II: Skills, allowed task families, and skill parameters.

Parameter	Relevant Skills	Search Space (Sim)	Search Space (Real)
(x, y) position	grasps, go-to pose	[-0.1, 0.1] x/y/z offset from object center	location on table surface
z-orientation	top grasp	$[0, 2\pi]$	$[0, 2\pi]$
approach angle	side grasp	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$
orientation	go-to pose	$[0, 2\pi]$ r/p/y Euler angles converted to quat	$[0, 2\pi]$ r/p/y Euler angles converted to quat
distance to lift	lift	$[0, 0.5]$	$[0, 0.5]$
rotation axis	rotate	[-0.1, 0.1] x/y offset from object center	location on table surface
rotation radius	rotate	$[0, 0.2]$	$[0, 0.2]$

TABLE III: Parameter search spaces for simulation and real world. In simulation, unlike the real world, we have access to object poses, so we can constrain some search spaces for efficiency. Positions, distances, and lengths are in meters. *Rotation axis* is always vertical.

0.5, gradient clip threshold 0.5, number of steps 10, number of minibatches per update 4, and number of optimization epochs 4. Our implementation builds on the Surreal Robotics Suite [32]. Training is parallelized across 50 workers. The time horizon $T = 3$ in all tasks.

B. Real-World Sawyer Experimental Setup

Environment. Our real-world setup also contains two Sawyer robot arms with parallel-jaw grippers placed at opposing ends of a table, facing each other. We task the robots with manipulating nine common household objects that require two parallel-jaw grippers to interact with. We consider the same four task families (lateral lifting, picking, opening and rotating), but work with more diverse objects (such as a rolling pin, soccer ball, glass jar, and T-wrench), as detailed in Table I. For each task family, we use the schema discovered for that family in simulation, and only learn the continuous parameterizations of the skills in the real world. See Figure 1 for pictures of some of our tasks.

Skills. The skills and parameters are the same as in simulation (Table II), but the search spaces are less constrained (Table III) since we do not have access to object poses.

State and Policy Representation. The state for these real-world tasks is the 256×256 RGB image obtained from an overhead camera that faces directly down at the table. To predict the continuous arguments, we use a fully convolutional spatial neural network architecture [33], as shown in Figure 4 along with example response maps.

Training Details. We use PPO and mostly the same hyper-parameters, with the following differences: learning rate

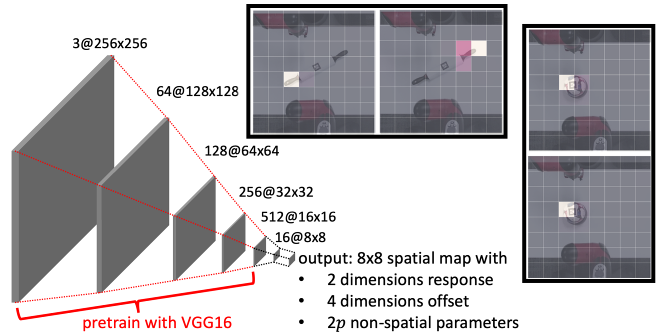

Fig. 4: To predict continuous arguments given an image input, we leverage the fact that all tasks require learning two spatial arguments for each arm: an (x, y) location along the table surface. To learn efficiently, we use a fully convolutional architecture [33]. We begin by passing the image through the first four layers of an ImageNet-pretrained VGG16 [34]; each layer is 2 convolutions followed by a max-pool. This gives us 512 8×8 maps. The second-last layer convolves with 16 2×2 filters with stride 2, giving 16 8×8 maps. Finally, the last layer convolves with $(2 + 4 + 2p)$ filters with stride 1, where p is the number of non-spatial parameters for the task. The first two dimensions give each arm's probability distribution (response map) over an 8×8 discretization of the table surface locations, the next four dimensions give offsets on these locations, and the final $2p$ dimensions give values for each arm's remaining arguments (orientations, distances, etc.). *Upper right:* Example response maps learned by this model for each arm, for the rolling pin and soccer ball tasks. The AR tags are only used in our experiments for automating the reward computation to measure success; our model is not given any object pose information.

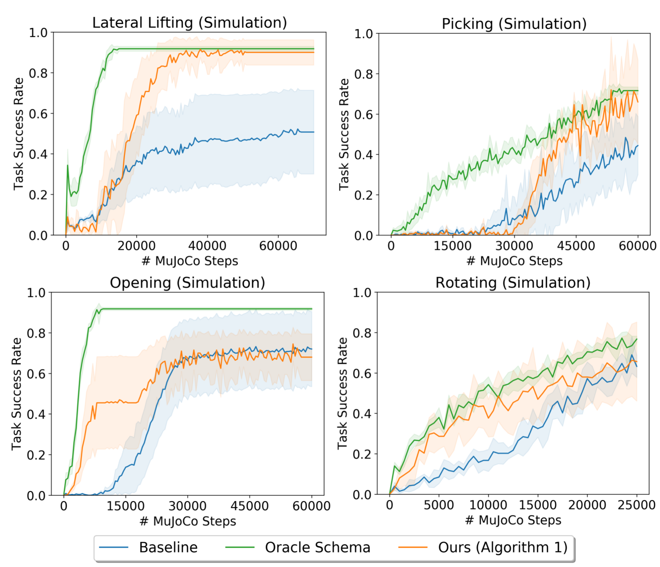

Fig. 5: Learning curves for simulated tasks, with training parallelized across 50 workers. Each curve depicts an average across 5 random seeds. By using a policy architecture that leverages the state-independence of the optimal schema (orange), we are able to achieve significant gains in sample complexity across all four tasks, over the baseline architecture that predicts skills and arguments conditioned on the state (blue). If an oracle tells us the perfect schema, and we only need to learn the arguments for those skills, then of course, learning will be extremely sample-efficient (green).

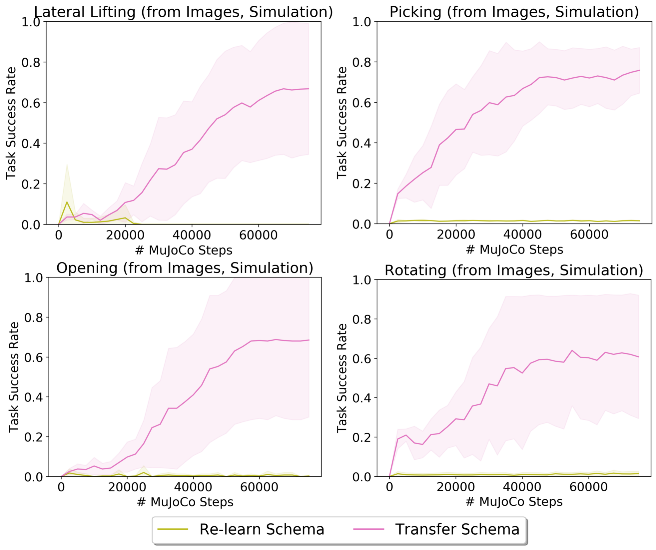

Fig. 6: Learning curves for training from images in simulation, with training parallelized across 50 workers. Each curve depicts an average across 5 random seeds. When learning visual policies, transferring schemas trained on low-dimensional states is crucial.

0.005, number of steps 500, number of minibatches per update 10, number of optimization epochs 10, and no parallelization. We control the Sawyers using PyRobot [35].

C. Results in Simulation

Figure 5 shows that our policy architecture greatly improves the sample efficiency of model-free reinforcement learning. In all simulated environments, our method learns the optimal schema, as shown in the last column of Table I. Much of the difficulty in these tasks stems from sequencing the skills correctly, and so our method, which more effectively shares experience across task instantiations in its attempt to learn the task schema, performs very well.

Before transferring the learned schemas to the real-world tasks, we consider learning from rendered images in simulation, using the architecture from Figure 4 to process them. Figure 6 shows the impact of transferring the schema versus re-learning it in this more realistic simulation setting. We see that when learning visual policies, transferring the schemas learned in the tasks with low-dimensional state spaces is critical to efficient training. These results increase our confidence that transferring the schema will enable efficient real-world training with raw RGB images, as we show next.

D. Results in Real World

Figure 7 shows our results on the nine real-world tasks, with schemas transferred from the simulated tasks. We can see that, despite the challenging nature of the problem (learning from raw camera images, given sparse rewards), our system is able to learn to manipulate most objects in around 4-10 hours of training. We believe that our approach can be useful for sample-efficient learning in problems other than manipulation as well; all one needs is to define skills appropriate for the environment such that the optimal sequence

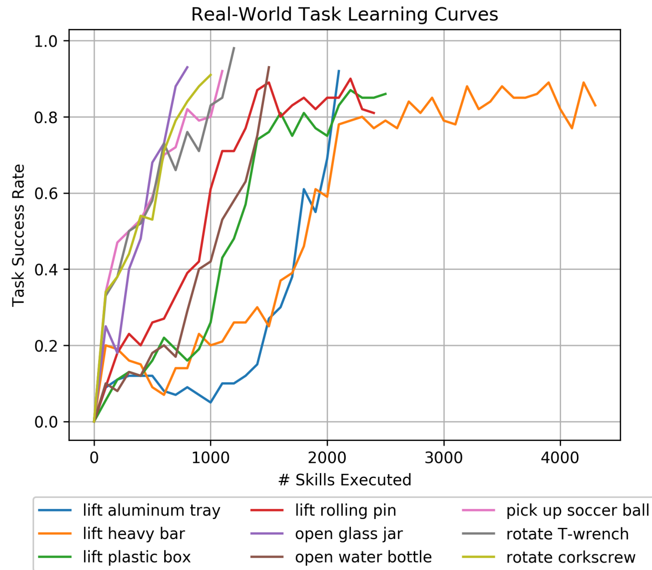

Fig. 7: Learning curves for the real-world tasks. We stop training when the policy reaches 90% average success rate over the last 100 episodes. It takes around 6-8 hours to execute 2000 skills, so most policies are learned in around 4-10 hours. By transferring schemas learned in simulation, we can train robots to solve sparse-reward bimanual manipulation tasks from raw camera images.

depends only on the task, not the (dynamic) state. The skills may themselves be parameterized closed-loop policies.

Please see the supplementary video for examples of learned behavior on the real-world tasks.

V. Future Work

In this work, we have studied how to leverage state-independent sequences of skills to greatly improve the sample efficiency of model-free reinforcement learning. Furthermore, we have shown experimentally that transferring sequences of skills learned in simulation to real-world tasks enables us to solve sparse-reward problems from images very efficiently, making it feasible to train real robots to perform complex skills such as bimanual manipulation.

An important avenue for future work is to relax the assumption that the optimal schema is open-loop. For instance, one could imagine predicting the schema via a recurrent mechanism, so that the decision on what skill to use at time $t+1$ is conditioned on the skill used at time t. Another interesting future direction is to study alternative approaches to training the state-independent schema predictor. Finally, we hope to investigate the idea of inferring the schemas from human demonstrations of a task, such as those in videos.

Acknowledgments

We thank Dhiraj Gandhi for help with the experimental setup. Rohan is supported by an NSF Graduate Research Fellowship through his capacity with the Massachusetts Institute of Technology. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[5] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[7] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[8] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[10] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[11] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," 2016.

[12] E. Wei, D. Wicke, and S. Luke, "Hierarchical approaches for reinforcement learning in parameterized action space," in *2018 AAAI Spring Symposium Series*, 2018.

[13] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 166–175.

[14] Y. Davidor, *Genetic Algorithms and Robotics: A heuristic strategy for optimization*. World Scientific, 1991, vol. 1.

[15] E. Gat, "On the role of simulation in the study of autonomous mobile robots," in *AAAI-95 Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*, 1995.

[16] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5307–5314.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[18] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[19] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *Robotics: Science and Systems*, 2017.

[20] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive behavior acquisition for a real robot by vision-based reinforcement learning," *Machine learning*, vol. 23, no. 2-3, pp. 279–303, 1996.

[21] L. Chrisman, "Reasoning about probabilistic actions at multiple levels of granularity," in *AAAI Spring Symposium: Decision-Theoretic Planning*, 1994.

[22] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in neural information processing systems*, 1993, pp. 271–278.

[23] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier, "Hierarchical solution of Markov decision processes using macro-actions," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 220–229.

[24] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," in *International Conference on Learning Representations*, 2018.

[25] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3303–3313.

[26] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[27] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[29] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[30] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

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

[32] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, "SURREAL: Open-source reinforcement learning framework and robot manipulation benchmark," in *Conference on Robot Learning*, 2018.

[33] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[35] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta, "Pyrobot: An open-source robotics framework for research and benchmarking," *arXiv preprint arXiv:1906.08236*, 2019.