2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Mastering Stacking of Diverse Shapes with Large-Scale Iterative Reinforcement Learning on Real Robots

Thomas Lampe*, Abbas Abdolmaleki*, Sarah Bechtle*, Sandy H. Huang*, Jost Tobias Springenberg*, Michael Bloesch, Oliver Groth, Roland Hafner, Tim Hertweck, Michael Neunert, Markus Wulfmeier, Jingwei Zhang, Francesco Nori, Nicolas Heess, Martin Riedmiller

Abstract— **Reinforcement learning solely from an agent's self-generated data is often believed to be infeasible for learning on real robots, due to the amount of data needed. However, if done right, agents learning from real data can be surprisingly efficient through re-using previously collected sub-optimal data. In this paper we demonstrate how the increased understanding of off-policy learning methods and their embedding in an iterative online/offline scheme ("collect and infer") can drastically improve data-efficiency by using all the collected experience, which empowers learning from real robot experience only. Moreover, the resulting policy improves significantly over the state of the art on a recently proposed real robot manipulation benchmark. Our approach learns end-to-end, directly from pixels, and does not rely on additional human domain knowledge such as a simulator or demonstrations.**

I. Introduction

Recent years have seen significant progress in learning based approaches for the control of real robots. Notably, reinforcement learning has been used to produce high-quality controllers in simulation that can then be transferred to the real world with sim2real approaches [1], [2], [3], [4], [5], [6], [7], while learning from demonstrations in the form of teleoperated robot trajectories (behavior cloning) directly in the real world has been shown to be surprisingly data-efficient and effective [8], [9], [10], [11].

In contrast, reinforcement learning (RL) directly on real robots has received comparatively less attention, despite several attractive properties. Unlike behavior cloning (BC), RL does not rely on expert data. Therefore it is not bounded by the performance of a human teleoperator, does not require potentially expensive teleoperation rigs, and is thus also applicable to robots that cannot be effectively teleoperated. In addition, unlike sim2real, learning directly on real robots does not require a simulator that is carefully matched to reality. Thus it can directly take advantage of the diversity of the real world, and, for instance, of sensors that are available on real robots but hard to simulate.

There are several reasons why RL on real robots has received less attention in recent years. These include questions of data-efficiency, the difficulty of establishing a safe and effective data collection scheme, and also the difficulty of algorithm tuning and of making suitable hyper-parameter choices that avoid instabilities or premature convergence, especially in an online setting.

*Primary authors; correspondence: THOMASLAMPE@GOOGLE.COM. All authors affiliated with Google DeepMind, London N1C4AG

However, RL algorithms have improved considerably during the last decade. In particular, modern variants of off-policy and offline algorithms are considerably more data-efficient and less sensitive to parameter choices than their predecessors. Importantly, they also allow for a variety of learning scenarios where data collection and policy optimization are interleaved in flexible ways. This makes it possible, for instance, to start and stop data collection as needed, to reuse data from prior experiments and mix different data sources (like self-generated and expert data, or data from different policies), and also to test different algorithm variants and hyper-parameter choices with the same set of data.

This flexibility and its implications have been highlighted by the "Collect-and-Infer" paradigm [12], which emphasizes the idea that data collection and policy optimization are two distinct processes that can be optimized separately. In this paper we take inspiration from Collect and Infer, and explore how the aforementioned flexibility can be used to create practical and surprisingly robust and data-efficient iterative schemes for real-robot RL.

As a test case we consider RGB Stacking [4], a robot manipulation benchmark that involves stacking of geometric shapes, which emphasizes precision and the understanding of geometric affordances. The contact dynamics of this task are non-trivial to replicate in simulation, thus making it an interesting test case for real robot learning.

Our approach consists of two pairs of online/offline stages, where we collect data during the online phase and subsequently perform offline policy optimization. In the first iteration we (1) perform online off-policy RL to learn an initial policy and collect a diverse dataset. This is followed by (2) a first round of offline policy optimization during which we explore different algorithm settings and model architectures. (3) In the online phase of the second iteration we evaluate the policies from (2), which sheds light on suitable hyperparameter settings and collects additional, higher-quality data. (4) The final policy is then obtained through offline policy optimization using the full dataset collected so far.

This scheme provides significant flexibility in each phase. For instance, this allows multi-task training for exploration, efficient exploration of different algorithm parameters, and the ability to restart learning with a different network architecture, thus avoiding premature convergence. Overall, this results in a robust and data-efficient learning scheme. Furthermore, training directly on real data removes the need for expensive

979-8-3503-8457-4/24/$31.00 ©2024 IEEE

tuning of a simulator, and allows the policy to take advantage of sensors not available in simulation. The final policy solves the task near flawlessly, outperforming previously published results on this benchmark by a large margin.

II. Background

A. Markov Decision Processes

We consider the problem setting of a Markov Decision Process (MDP), defined by a state space $\mathcal{S}$, action space $\mathcal{A}$, transition model $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$, reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, discount factor $\gamma \in [0, 1)$, and initial state distribution μ_0. A policy $\pi_\theta(a|s, k)$ specifies a distribution over actions $a \in \mathcal{A}$, given a state $s \in \mathcal{S}$ and task identifier $k \in [1, N]$. The action-value function, or Q-function, for a policy π_θ is the expected return from taking action a in state s and then following this policy: $Q^\pi(s, a; k) = \mathbb{E}_{s' \sim \mathcal{P}(s,a)}[r_k(s, a) + \mathbb{E}_{a' \sim \pi(s')}[Q^\pi(s', a'; k)]]$.

B. Multi-task RL and Scheduled Auxiliary Control (SAC-X)

In the multi-task RL setting, there are N tasks, each of which has a corresponding reward function r_k. In this work, we use multi-task RL for both the initial data collection phase and for offline RL, to gather more diverse data and stabilize learning, respectively.

In particular, we use Scheduled Auxiliary Control (SAC-X) [13], which is a multi-task off-policy actor-critic algorithm. The policy and Q-function networks each have a shared torso across tasks, with a separate output head per task. SAC-X is a hierarchical agent, where data is gathered by a scheduler choosing which of the task policies to execute. In the SAC-Q variant, the scheduler chooses tasks to execute in order to maximize the reward of a predefined main task.

SAC-X trains policies with multi-task policy iteration, alternating between policy evaluation (i.e., updating the Q-functions) and policy improvement (i.e., updating the policies). In the policy evaluation step, for each task k, given a batch of transitions $\{s, a, s', \{r_k\}_{k=1}^N\}$ and the current per-task policy π_k^{old}, any policy evaluation algorithm can be used to update the corresponding per-task Q-function Q_k. In this work, we use n-step return combined with distributional Q-learning [14].

For the policy improvement step, SAC-X uses either Stochastic Value Gradients [15] or maximum a posteriori policy optimization (MPO) [16]. In this work we use MPO for this step, although in theory any policy improvement algorithm could be used, e.g. Soft Actor Critic [17].

III. Method

Our approach, dubbed CHEF, is inspired by the Collect and Infer paradigm introduced by [12]. It consists of four stages:

1) **C**ollect a real-world dataset
2) **H**yperparameter exploration: offline RL to train policies
3) **E**xecute policies on real robots, and collect this data
4) **F**ine-tune policies on all collected data

In this framework, no data is wasted—in Stages 1 and 3, we collect real-world data, while in Stages 2 and 4, we train policies on the collected data. No new data is collected in Stages 2 and 4. In Stage 3, we evaluate policies with no additional training; this is necessary regardless for evaluating the performance of policies trained fully offline.

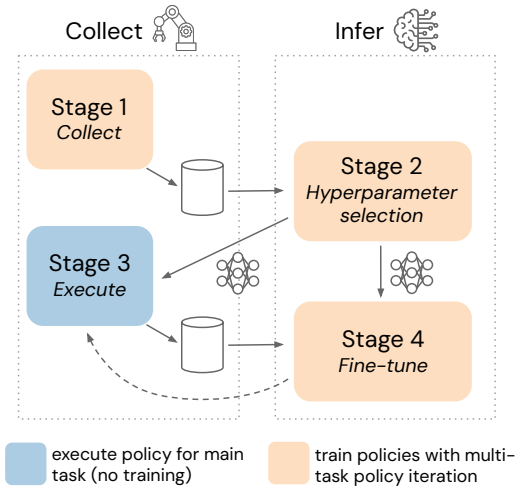


Fig. 1: Illustration of our approach, CHEF, and how it relates to the more general collect-and-infer paradigm. We collect data in Stages 1 and 3, and train policies with offline RL on the data in Stages 2 and 4.

Collect (Stage 1). We use SAC-Q to train multi-task policies from scratch on real robots. The SAC-Q algorithm uses the scheduler to decide which sequence of task policies to execute in each episode. By optimizing policies for different tasks and sequencing them in multiple ways, this algorithm achieves better coverage of the state space compared to single task training, and thus collects to a more diverse dataset. We continue Stage 1 until the policies for all tasks have converged (albeit to a sub-optimal solution), and store all interaction data for future use.

Hyperparameter exploration (Stage 2). In Stage 2, we train agents in an offline RL setting where the transitions come from the dataset collected in Stage 1. We collect no new data in this phase. We sweep over a variety of hyperparameters, including network architecture and training parameters. We train agents with multi-task policy iteration; this is exactly the same as in Stage 1, except now the transitions come from a fixed dataset rather than from executing the scheduler on a real robot. This works surprisingly well, given that in general off-policy RL algorithms cannot be directly used for offline RL, due to extrapolation error [18]. In contrast, we find that multi-task training with a suitable architecture sufficiently stabilizes offline learning when data-coverage of the state-action space is good (as encouraged by our multi-task data collection). We investigate this further with ablations in Section VII-E.

Execute (Stage 3). We execute fully-trained policies from each of the hyperparameter choices in Stage 2, on real robots, and save all the data gathered. This allows us not only to determine which of the hyperparameters results in the best policy, but also to gather extra high-quality data. This minimizes the additional data necessary for hyperparameter selection. In contrast, naïve hyperparameter selection would require either training from scratch for each

choice of hyperparameters, or selecting hyperparameters based on what works best in simulation, and hoping that those work on the real system.

FINE-TUNE (Stage 4). We then use all of the data gathered so far (from Stages 1 and 3) to fine-tune the best policies offline, based on the real-world evaluations from Stage 3.

IV. RELATED WORK

A. Offline RL for robot learning

Offline RL is the data-driven formulation of the reinforcement learning problem. The aim is still to maximize reward; however, the agent can no longer interact with the environment and collect additional transitions [19], [20]. Recent actor-critic algorithms for offline RL include Critic Regularized Regression [21] for simulated continuous control tasks; conservative Q-learning [22], which learns a lower bound on the policy value for stabilizing offline RL; and QT-Opt [23], a self-supervised vision-based framework. More recently in [24], goal-conditioned offline Q-learning is used to learn robotic skills of discrete actions from previously collected offline data without access to specified rewards; this enables learning a variety of skills on real robots. Beyond actor-critic algorithms, in both [25] and [26], a reward model is learned from human preferences to annotate existing offline datasets, which is then used to perform offline RL. In [27] a teacher student learning setup is presented where a dataset collected by a suboptimal teacher is used for offline RL to warm-start the student policy. In this work we present an offline RL algorithm that learns an actor and critic in a multi-task fashion, alternating between online data collection and offline learning to master the RGB Stacking task from [4] and used in [27].

B. Multi-task RL for robot learning

Multi-task RL holds the promise of amortizing the cost of single task learning by providing a shared representation across tasks. An approach to multi-task RL is to condition policies on tasks [28], [13], [29] or distill single task policies into a shared multi-task policy [30], [31], [32], [33], [34].

Other works create a mapping between tasks and individual policy parameters, in order to select the adequate policy for the specific task at hand [35], [36]. All these approaches present fairly hierarchical setups to solving the multi-task RL problem. In [37] the authors show that scaling up multi task RL with real robot data enables transfer and even zero-shot generalization. Similarly, this work focuses also on using real robot data for actor-critic learning; however, our work aims to minimize the amount of real robot data needed, by training policies with offline multi-task RL.

V. BENCHMARK DESCRIPTION

Block stacking has long been a standard benchmark task for robotic manipulation, with early work on vision-based block stacking from [28]. Recent works have tackled this problem by learning a curriculum of the different stages of the task [13], combining human demonstrations and RL [1], [25], enabling sim-to-real transfer [1], [2], [3], [4], and learning a generalist transformer-based policy from expert demonstrations [11].

The RGB Stacking benchmark we consider in this work was first presented in [4], [27]. The task setup consists of 5 distinct block stacking configurations, that consist of parameterized geometric shapes of red, green and blue objects. This is a challenging benchmark for robotic manipulation. In contrast to our work that learns to master stacking purely from data collected on the real robot, the authors in [4] adopt a sim-to-real approach with a final fine-tuning stage on hardware. The manipulator used in this work is the Rethink Sawyer robot [38], which is controlled here in task space, with a 5D action consisting of the velocities of the end-effector's translation in Cartesian space, the wrist rotation, and the attached parallel gripper. It is kept within virtual walls to ensure safe autonomous operation. The policy has access to a mixture of proprioceptive observations (positions, velocities and forces of the end effector and joints) and 3 low-resolution (128x128) images from stationary cameras attached to the workspace. The observations notably exclude the positions of the objects; while those are tracked for the purpose of automation and reward computation, the agent must learn to act from vision alone.

The role of the shapes is color-coded: the red object should be stacked on the blue one, and a third green one serves as a distractor. Due to the different shapes of the objects, an agent is forced to learn about various geometric properties, including slanted faces, lengths, and off-center balancing.

This task remains challenging for existing methods to master, especially because success depends both on the contact physics of objects and on force interactions, e.g. when objects are not oriented in a way that would enable stacking, and need to be carefully nudged onto their side. Both aspects can prove difficult to model in simulation with sufficient fidelity to enable transfer, and are also hard to demonstrate via teleoperation due to limited force feedback.

In this work, we are specifically considering the "skill mastery" sub-task, where the goal is to achieve maximum performance with a single policy on five test object triplets (Fig. 2b), and the test objects are available during training.

VI. EXPERIMENT SETUP

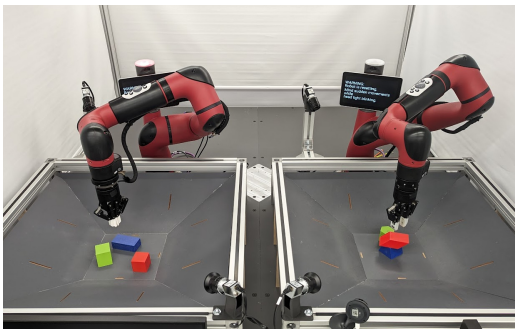
A. Sub-tasks

The SAC-Q multi-task learning algorithm described in Section II-B requires defining a set of sub-tasks. For the sub-task rewards we use the components of the composite reward in [4]. We choose rewards that form a natural curriculum from simpler to more complex multi-stage behaviors for better data efficiency, although more general rewards can also be used in principle [39]. We describe them conceptually below; for a full description including mathematical formulations, see [4].

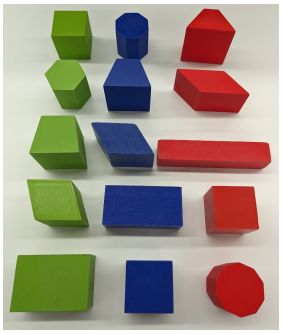
open Proportional to the opening angle of the gripper.

reach-grasp Sum of a shaped component for moving close to the red object, and a sparse component for triggering the gripper's contact detection.

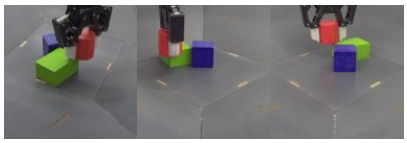
lift Shaped reward, proportional to the red object's height.



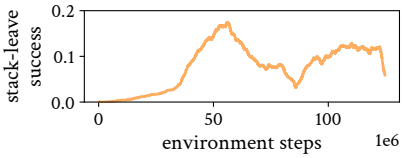
(a) Robot setups used in the RGB Stacking benchmark, showing the Sawyer arm, basket, and attached cameras.



(b) Triplets 1 to 5 (top to bottom) in the skill mastery challenge.



(c) Cropped, scaled images given to the agent.



(d) Average **stack-leave** success in COLLECT.

Fig. 2: The robot, objects, and agent inputs comprising the RGB Stacking benchmark, and learning in the COLLECT stage.

place Shaped reward, decreases non-linearly as the red object approaches a position directly above the blue object.

stack Sparse reward, non-zero when the red object is on top of the blue object, with a small tolerance.

stack-leave Sparse reward, non-zero when the conditions for **stack** hold, and the tool center point of the gripper is at least $10cm$ from the red object's center of mass.

B. CHEF *Stages*

As per the CHEF approach in Section III, we perform training in distinct stages, concretely performed as follows:

COLLECT: Online RL for approximately 330k episodes. This stage is distributed across a fleet of identical setups, with 10 robots collecting episodes, writing their data to a shared experience replay, and retrieving the newest online-trained policy after each episode. We train the policies until the task return converges (Fig. 2d).

HYPERPARAMETER EXPLORATION: We vary several hyperparameters, described in Section VII. Candidate policies are trained with offline RL for 2M update steps each, at which point performance has reliably converged.

EXECUTE: Evaluating the policies generated in the previous stage results in approximately 70k additional episodes. All of these evaluations are performed for the **stack-leave** sub-task only, even for multi-task policies.

FINE-TUNE: We continue training the best policy from the previous stage for another 2M steps, using the combined 400k episodes from both data collection phases.

C. Model Architecture

For both the critic and the actor network, we use a Residual Neural Network [40] (ResNet) for the shared torso, with a separate MLP head per task. Details are provided in the supplementary material[1]. Most parameter settings are shared across all experiments, and are chosen based on prior simulation experiments. The most influential parameter we vary is the number and size of ResNet channels; these will be compared in the results below.

[1]https://sites.google.com/view/robochef

Method	Triplet					Average
	1	2	3	4	5	
BC (S2R + R) [4]	76%	61%	71%	88%	78%	75%
CRR (S2R + R) [4]	87%	68%	75%	88%	89%	82%
R-MPO (S2R + R) [27]	82%	55%	73%	92%	89%	78%
R-CRR (S2R + R) [27]	54%	61%	74%	94%	94%	75%
COLLECT	6%	22%	27%	47%	29%	26%
HYPERPARAM.	92%	89%	93%	92%	97%	92%
EXECUTE	62%	50%	63%	74%	71%	61%
FINE-TUNE	**96%**	**97%**	**95%**	**95%**	**98%**	**96%**

TABLE I: Comparison between prior work and the four CHEF stages. Performance for HYPERPARAMETER EXPLORATION is for the best policy, chosen for fine-tuning later. Performance for EXECUTE is the average of all evaluations. For baselines, S2R denotes sim-to-real and R denotes real-data used.

D. Evaluation

In the EXECUTE stage, we evaluate each candidate policy for 200 episodes. Between episodes, a hand-written controller randomizes the positions of all objects in the workspace, and moves the end effector to a random position.

As a performance metric, we report the percentage of episodes deemed "successful", defined by the **stack-leave** reward component being 1 at the end of the episode. Each episode lasts for 20 seconds, at a control rate of 20 Hz. Episodes are terminated prematurely if the wrist force-torque sensor registers a force above $20N$ at any time.

VII. RESULTS

We first present the final performance of the policy trained through our approach, compared to state-of-the-art baselines (Table I). A single iteration of offline RL is in fact already sufficient to outperform the baselines (92% vs 82% for the strongest baseline). This is a remarkably large improvement, compared to the end of the COLLECT stage (26%). We attribute this large jump to the fact that the algorithm now has access to the full experience from the start of training. Additionally, no care needs to be taken to avoid overfitting when the data distribution is still narrow, so the size of the model can be increased. Nonetheless, performance after this first offline phase is not optimal. Only after the second round of collect-and-infer does it reach near-optimality (96%).

In the remainder of this section, we will discuss the three design choices we investigated in the HYPERPARAMETER EXPLORATION stage: inclusion of real-world-only observations, multi-task offline learning, and network size. We will also present several ablations, on how the final FINE-TUNE policy is trained and how the COLLECT and HYPERPARAMETER EXPLORATION stages are designed.

A. Hyperparameter Exploration: Observations

One of the main motivations for learning purely from real data, rather than also leveraging simulators, is the ability to rely on sensor data that is difficult to simulate. In particular, good performance on some test triplets involves gentle interaction with the objects. For Triplet 2, the blue object may need to be flipped over, as illustrated in Fig. 3. This must be done without exceeding wrist force-torque thresholds, which would trigger an episode termination. That in turn requires the use of the robot's wrist force sensor, which was left out in previous work because the large sim-to-real gap of these measurements was found to hinder transfer.

Using only simulation-capable observations leads to an average success rate of 43% on the relevant Triplet 2. In contrast, including haptic observations, namely forces and torques for both wrist and joint sensors, significantly increases the success rate to 78% after the first round of offline RL, with all other settings being equal. Qualitatively, this leads to more deliberate policies that carefully nudge the bottom object when necessary, and reliably avoid force-torque-based early terminations. For Triplet 2, the percentage of episodes terminated early drops from 30.2% without haptic observations, to 5.6% when including them. Examples of this behaviour are provided in the supplementary video.

B. Hyperparameter Exploration: Network Size

An advantage of offline learning is that it enables us to switch the network architecture after the initial COLLECT phase. For collection, it can be preferable to use a smaller model, to reduce the risk of overfitting and increase the speed of parameter updates: we used a ResNet with channel sizes of $\{16, 32, 32\}$ and an embedding size of 32. However, for the final offline policy, a larger model is preferable, to ensure that the policy can be fit with maximum precision. Increasing the channel sizes to $\{64, 128, 128, 64\}$ and the embedding size to 256 yields a higher success rate (Table II-B).

C. Hyperparameter Exploration: Multi-task Offline Learning

The performance of our approach relies on the use of multi-task offline RL, together with non-expert data with good coverage [41]. For the multi-task setup, we use an SAC-Q multi-task critic as described in Section II-B, with the same six sub-tasks as used during the initial COLLECT stage. Single-task offline RL for only the **stack-leave** reward performs substantially worse than multi-task (Table II-C); in this comparison both use the smaller network architecture.

Data coverage is also essential for offline RL: if we filter the data to include only successful episodes, then performance drops from 86% to 11%, because the critic overestimates the value of non-covered states and actions. However, for behavior cloning (BC) in our setup, this data filtering is necessary because the COLLECT dataset contains many unsuccessful episodes. Compared to filtered BC, multi-task offline RL performs better because it learns to maximize reward across the whole state-action distribution covered by the non-expert data, whereas BC suffers from compounding errors [42].

	Method	Triplet 1	2	3	4	5	Average
VII-B	Small network	87%	80%	86%	89%	91%	86%
	Large network	**92%**	**89%**	**93%**	**92%**	**97%**	**92%**
VII-C	Single-task RL	51%	75%	60%	**90%**	**93%**	74%
	Multi-task RL	**87%**	**80%**	**86%**	89%	91%	**86%**
	w/ filtered data	1%	3%	7%	27%	13%	11%
	Filtered BC	78%	60%	48%	80%	91%	71%
VII-D	Iteration 1	91%	73%	76%	92%	91%	84%
	Fine-tuning 2M	**97%**	**90%**	**94%**	**96%**	98%	**95%**
	Re-training 2M	93%	78%	91%	**96%**	**99%**	91%
	Re-training 4M	96%	87%	93%	**96%**	98%	94%

TABLE II: Success rates for ablations of our approach.

D. Ablation: Fine-tuning

For the final stage, we fine-tune the best policy, based on the results from the EXECUTE stage. Every mini-batch consists equally of samples drawn from the non-expert, multi-task dataset from the COLLECT stage, and the more success-heavy, single-task dataset produced during the EXECUTE stage.

Compared to training from scratch, fine-tuning reaches similar performance in half as many learner updates – 2M as opposed to 4M (Table II-D). When training from scratch for only 2M updates, performance is noticeably lower on Triplet 2. To put the reduction of required compute into scale, it takes approximately 18 days to train the large network for 2M updates, on 16 Google Cloud TPUv3 accelerators.

E. Ablations in Simulation

While we did not use simulation for pre-training or transfer, we did rely on it to inform the design of the COLLECT and HYPERPARAMETER EXPLORATION stages.

A key component of our approach is sharing the critic network torso across tasks. In simulation, we ablate this for the same RGB stacking benchmark, but with state features instead of image inputs. In the online RL setting, which is analogous to COLLECT, we found that sharing the critic torso leads to both faster learning and better performance for the main **stack-leave** task, whereas sharing the actor torso is less crucial (Fig. 4a, right). Note that much of the data in the experience replay is off-policy with respect to an individual task's policy, since during data collection, the scheduler sequences all the task policies together. Sharing the critic torso enables learning from this off-policy data, as shown by the shorter delay between when the data collection policy obtains non-zero reward for **stack-leave** (Fig. 4a, left) and when the task policy starts learning from this data. We hypothesize that sharing the critic torso enables learning a shared representation that simplifies critic learning for harder, sparse-reward tasks like

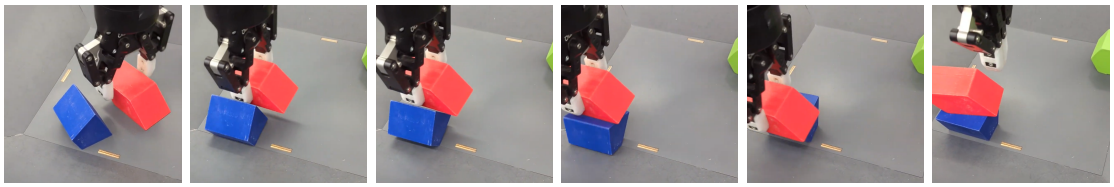
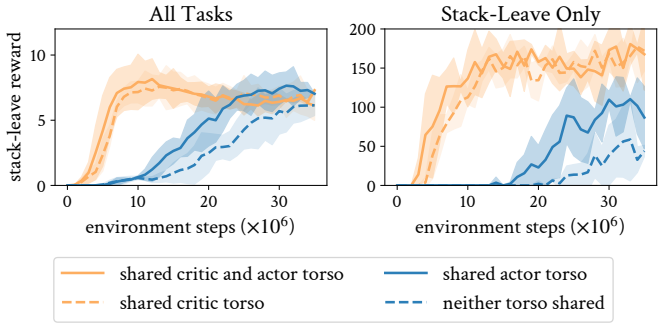
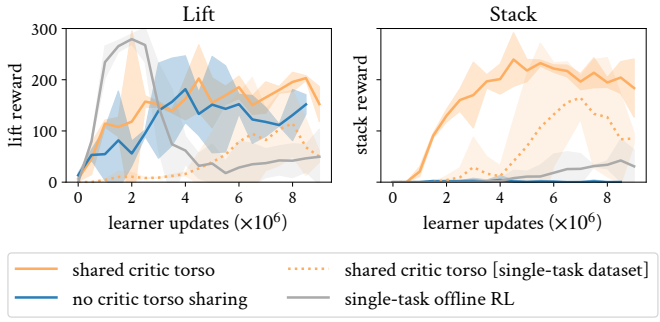


Fig. 3: Successful stacking attempt for triplet 2. The agent first aligns the blue object with the basket's edge and then carefully pushes against it in order to flip it into an orientation that the red object can be stacked onto.



(a) Online RL, analogous to COLLECT. The reward for **stack-leave** across training, for the data collection policy (left) and the task policy (right). Sharing the critic torso improves sample efficiency and final performance. When the critic torso is not shared, there is a delay from when the data collection starts obtaining reward for **stack-leave** after around 5M environment steps, compared to when the **stack-leave** policy learns from this data, more than 5M steps later.



(b) Offline RL, analogous to HYPERPARAMETER EXPLORATION. Plots show performance of the **lift** and **stack** task policies. Sharing the critic network torso across tasks does not impact performance on **lift**, but is essential for learning **stack**, a harder and sparse-reward task. The multi-task setup is important for both gathering data (i.e., COLLECT) and learning from this data, as shown by the worse performance on **stack** (orange dotted and grey lines, respectively).

Fig. 4: Ablations in simulation, on sharing network torsos during online and offline learning, and how the dataset is gathered.

stack-leave. We found that when the critic torso is not shared, the critic for **stack-leave** overestimates the return.

In the offline RL setting, analogous to HYPERPARAMETER EXPLORATION, sharing the critic network torso is even more important. Here we focus on a subset of four tasks: **reach-grasp**, **lift**, **place**, and **stack**. We first gather 40k episodes with multi-task online RL, as in COLLECT. Without a shared critic torso, the agent learns the **lift** task (Fig. 4b, left) but cannot learn the sparse-reward **stack** task (Fig. 4b, right).

We also run two other ablations in this offline RL setting. First, we instead gather the dataset with single-task online RL, where the reward is the composite reward used in [4]. Second, instead of using multi-task offline RL, we train a policy to optimize the single composite reward. Both cannot learn to stack consistently. These ablations indicate the importance of using a multi-task setup. Gathering the dataset in a multi-task setup improves coverage and includes trajectories that perform well for each of the individual tasks. Training policies offline in a multi-task setup enables using the learning signal from easier tasks to simplify learning for harder tasks.

VIII. DISCUSSION

We presented the CHEF schema, targeted toward off-policy actor-critic reinforcement learning. By separating the learning into distinct stages for exploration-driven data collection, offline hyperparameter search, greedy execution, and offline fine-tuning, we achieved maximum reuse of collected data for improved data efficiency, while also maintaining flexibility with regards to architecture and hyperparameters.

We applied this schema to the RGB Stacking benchmark, on real robots. Our approach not only reduced the required amount of system interaction, but also achieved near-perfect success, significantly improving upon the state-of-the-art. Our ablations highlighted several key features of the approach: First, using end-to-end RL to train directly on real robots, without the need for a simulator, allowed the use of relevant sensors that were previously ignored due to limited fidelity in simulation (Section VII-A). Second, offline RL coupled with multi-task exploration was able to utilize the large quantity of non-successful data produced during early training, outperforming filtered BC (Section VII-C). In addition, using a multi-task critic network architecture was key for stabilizing offline RL (Sections VII-C, VII-E). Finally, by using fine-tuning in the second offline training stage, we achieved similar performance compared to training from scratch, with half the computational cost (Section VII-D).

In this work, we used a set of sub-tasks that form a curriculum towards the desired main task. In future work, we would like to explore how the selection of tasks affects these findings, in particular as large models are being used to train many tasks in many domains at once (e.g. [8], [9], [11]).

To solve the RGB Stacking benchmark task, we only needed to perform the EXECUTE and FINE-TUNE stages once. If required, we could also repeat these stages until the desired performance is obtained, while accumulating the collected data. Such *iterative Collect-and-Infer* would allows the algorithm to 'hill-climb' to high performance via offline-RL starting from initial data. Additionally, the question arises whether repeating the HYPERPARAMETER EXPLORATION stage is optimal, or whether additional exploration would be required. While the answer may depend on the specific task considered, therein lies the strength of the general Collect-and-Infer framework, to choose the specific stages as needed.

REFERENCES

[1] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, *et al.*, "Reinforcement and imitation learning for diverse visuomotor skills," *arXiv preprint arXiv:1802.09564*, 2018.

[2] R. Jeong, Y. Aytar, D. Khosid, Y. Zhou, J. Kay, T. Lampe, K. Bousmalis, and F. Nori, "Self-supervised sim-to-real adaptation for visual robotic manipulation," in *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 2718–2724.

[3] L. Hermann, M. Argus, A. Eitel, A. Amiranashvili, W. Burgard, and T. Brox, "Adaptive curriculum generation from demonstrations for sim-to-real visuomotor control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6498–6505.

[4] A. X. Lee, C. M. Devin, Y. Zhou, T. Lampe, K. Bousmalis, J. T. Springenberg, A. Byravan, A. Abdolmaleki, N. Gileadi, D. Khosid, *et al.*, "Beyond pick-and-place: Tackling robotic stacking of diverse shapes," in *5th Annual Conference on Robot Learning*, 2021.

[5] T. Haarnoja, B. Moran, G. Lever, S. H. Huang, D. Tirumala, M. Wulfmeier, J. Humplik, S. Tunyasuvunakool, N. Y. Siegel, R. Hafner, *et al.*, "Learning agile soccer skills for a bipedal robot with deep reinforcement learning," *arXiv preprint arXiv:2304.13653*, 2023.

[6] S. Bohez, S. Tunyasuvunakool, P. Brakel, F. Sadeghi, L. Hasenclever, Y. Tassa, E. Parisotto, J. Humplik, T. Haarnoja, R. Hafner, *et al.*, "Imitate and repurpose: Learning reusable robot movement skills from human and animal behaviors," *arXiv preprint arXiv:2203.17138*, 2022.

[7] L. Smith, J. C. Kew, T. Li, L. Luu, X. B. Peng, S. Ha, J. Tan, and S. Levine, "Learning and adapting agile locomotion skills by transferring experience," *arXiv preprint arXiv:2304.09834*, 2023.

[8] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022.

[9] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint arXiv:2307.15818*, 2023.

[10] A. Stone, T. Xiao, Y. Lu, K. Gopalakrishnan, K.-H. Lee, Q. Vuong, P. Wohlhart, B. Zitkovich, F. Xia, C. Finn, *et al.*, "Open-world object manipulation using pre-trained vision-language models," *arXiv preprint arXiv:2303.00905*, 2023.

[11] K. Bousmalis, G. Vezzani, D. Rao, C. Devin, A. X. Lee, M. Bauza, T. Davchev, Y. Zhou, A. Gupta, A. Raju, A. Laurens, C. Fantacci, V. Dalibard, M. Zambelli, M. Martins, R. Pevceviciute, M. Blokzijl, M. Denil, N. Batchelor, T. Lampe, E. Parisotto, K. Żołna, S. Reed, S. G. Colmenarejo, J. Scholz, A. Abdolmaleki, O. Groth, J.-B. Regli, O. Sushkov, T. Rothörl, J. E. Chen, Y. Aytar, D. Barker, J. Ortiz, M. Riedmiller, J. T. Springenberg, R. Hadsell, F. Nori, and N. Heess, "Robocat: A self-improving foundation agent for robotic manipulation," 2023.

[12] M. Riedmiller, J. T. Springenberg, R. Hafner, and N. Heess, "Collect & infer – a fresh look at data-efficient reinforcement learnin," *arXiv preprint arXiv:2108.10273*, 2021.

[13] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing solving sparse reward tasks from scratch," in *International conference on machine learning*. PMLR, 2018, pp. 4344–4353.

[14] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International conference on machine learning*, 2017.

[15] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," *Advances in Neural Information Processing Systems*, 2015.

[16] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a Posteriori policy optimisation," in *International Conference on Learning Representations*, 2018.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, 2018.

[18] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*, 2019.

[19] S. Lange, T. Gabel, and M. Riedmiller, *Batch Reinforcement Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 45–73. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_2

[20] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review," *and Perspectives on Open Problems*, vol. 5, 2020.

[21] Z. Wang, A. Novikov, K. Zolna, J. S. Merel, J. T. Springenberg, S. E. Reed, B. Shahriari, N. Siegel, C. Gulcehre, N. Heess, *et al.*, "Critic regularized regression," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7768–7778, 2020.

[22] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[23] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *arXiv preprint arXiv:1806.10293*, 2018.

[24] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn, *et al.*, "Actionable models: Unsupervised offline reinforcement learning of robotic skills," *arXiv preprint arXiv:2104.07749*, 2021.

[25] S. Cabi, S. G. Colmenarejo, A. Novikov, K. Konyushkova, S. Reed, R. Jeong, K. Zolna, Y. Aytar, D. Budden, M. Vecerik, *et al.*, "Scaling data-driven robotics with reward sketching and batch reinforcement learning," *arXiv preprint arXiv:1909.12200*, 2019.

[26] D. Shin, A. Dragan, and D. S. Brown, "Benchmarks and algorithms for offline preference-based reward learning," *Transactions on Machine Learning Research*, 2023.

[27] A. X. Lee, C. Devin, J. T. Springenberg, Y. Zhou, T. Lampe, A. Abdolmaleki, and K. Bousmalis, "How to spend your robot time: Bridging kickstarting and offline reinforcement learning for vision-based robotic manipulation," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 2468–2475.

[28] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 3876–3881.

[29] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, N. Siegel, T. Hertweck, T. Lampe, N. Heess, and M. Riedmiller, "Compositional transfer in hierarchical reinforcement learning," *Robotics: Science and Systems XVI*, 2020.

[30] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[31] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[32] 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.

[33] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," *arXiv preprint arXiv:1511.06342*, 2015.

[34] D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar, and S. Levine, "Divide-and-conquer reinforcement learning," *arXiv preprint arXiv:1711.09874*, 2017.

[35] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, vol. 33, pp. 361–379, 2012.

[36] B. Da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," *arXiv preprint arXiv:1206.6398*, 2012.

[37] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman, "Scaling up multi-task robotic reinforcement learning," in *5th Annual Conference on Robot Learning*, 2021.

[38] R. Robotics, "Sawyer robot," https://www.rethinkrobotics.com/sawyer, 2023 (accessed September 1, 2023).

[39] T. Hertweck, M. A. Riedmiller, M. Bloesch, J. T. Springenberg, N. Y. Siegel, M. Wulfmeier, R. Hafner, and N. Heess, "Simple sensor intentions for exploration," *CoRR*, vol. abs/2005.07541, 2020. [Online]. Available: https://arxiv.org/abs/2005.07541

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[41] N. Lambert, M. Wulfmeier, W. Whitney, A. Byravan, M. Bloesch, V. Dasagi, T. Hertweck, and M. Riedmiller, "The challenges of exploration for offline reinforcement learning," *arXiv e-prints*, pp. arXiv–2201, 2022.

[42] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.