Fast Reinforcement Learning without Rewards or Demonstrations via Auxiliary Task Examples

Trevor Ablett[1], **Bryan Chan**[2], **Jayce Haoran Wang**[1], **Jonathan Kelly**[1]

[1]University of Toronto, [2]University of Alberta

Abstract:
An alternative to using hand-crafted rewards and full-trajectory demonstrations in reinforcement learning is the use of examples of completed tasks, but such an approach can be extremely sample inefficient. We introduce value-penalized auxiliary control from examples (VPACE), an algorithm that significantly improves exploration in example-based control by adding examples of simple auxiliary tasks and an above-success-level value penalty. Across both simulated and real robotic environments, we show that our approach substantially improves learning efficiency for challenging tasks, while maintaining bounded value estimates. Preliminary results also suggest that VPACE may learn more efficiently than the more common approaches of using full trajectories or true sparse rewards. Project site: https://papers.starslab.ca/vpace/.

1 Introduction

Example-based control (EBC), in which example states of completed tasks are used as feedback in reinforcement learning [1], can be far less laborious than designing a reward function or gathering trajectories. Unfortunately, excluding information on how goal states are reached can lead to highly inefficient learning (e.g., an example of a loaded dishwasher provides no information about the long sequence of actions required to complete the task). ***Can we speed up example-based control?***

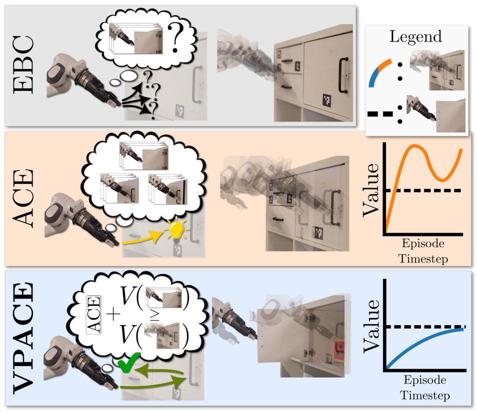


Figure 1: Example-based control (EBC) is inefficient due to poor exploration. Auxiliary control from examples (ACE) resolves this via exploration of auxiliary tasks, but can have unbounded value error. VPACE combines ACE with above-success-level value penalization (VP).

We introduce **v**alue-**p**enalized **a**uxiliary **c**ontrol from **e**xamples (**VPACE**), where we leverage the scheduled auxiliary control (SAC-X) [2] framework to introduce a set of simple and reusable auxiliary tasks, in addition to the main task, that help the agent to explore the environment. In this work, the full set of auxiliary tasks that we use across all main tasks are reach, grasp, lift, and release. For each auxiliary task there is a corresponding auxiliary policy that learns to match the set of examples. A scheduler periodically chooses and executes the different auxiliary policies, in addition to the main policy, generating a more diverse set of data to learn from.

We find that the naïve application of SAC-X to EBC can result in overestimated values, leading to sample inefficiency and poor performance. We present experiments across four environments with 19 simulated and two real tasks to show the improved sample efficiency and final performance of VPACE over EBC, inverse reinforcement learning, an exploration bonus, and the use of full trajectories and true sparse rewards.

Related Work. In inverse RL (IRL), a reward function is recovered from demonstrations, and a policy is learned either subsequently [3] or simultaneously via adversarial imitation learning (AIL) [4, 5, 6, 7]. In *example-based control* (EBC), an agent is only provided *successful example states* [8, 9, 10, 1]. Hierarchical reinforcement learning (HRL) aims to leverage multiple levels of abstraction in long-horizon tasks [11], improving exploration in RL [12, 13, 14]. Scheduled auxiliary control (SAC-X, [2]) combines a scheduler with semantically meaningful and simple auxiliary sparse rewards or auxiliary full expert trajectories (LfGP, [15, 16, 17]). [18] combined EBC with hierarchical learning, but their approach required a symbolic planner at test time. Regularization and clipping techniques have been applied to off-policy RL to address various problems such as stabilizing the bootstrapping target [19, 20] and preventing overfitting and out-of-distribution samples [21, 22].

2 Example-Based Control with Value-Penalization and Auxiliary Tasks

Problem Setting. In *example-based control* (EBC), we are given a set of example states of a completed task: $s^* \in \mathcal{B}^*$, where $\mathcal{B}^* \subseteq \mathcal{S}$ and $|\mathcal{B}^*| < \infty$. The goal is to (i) leverage $\mathcal{B}^*$ and $\mathcal{B}$ to learn or define a state-conditional reward function $\hat{R} : \mathcal{S} \to \mathbb{R}$ that satisfies $\hat{R}(s^*) \geq \hat{R}(s)$ for all $(s^*, s) \in \mathcal{B}^* \times \mathcal{B}$, and (ii) learn a policy $\hat{\pi}$ that maximizes the expected return $\hat{\pi} = \arg\max_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \hat{R}(s_t)\right]$, where γ is a discount factor.

For any policy π, we can define the value function and Q-function respectively to be $V^\pi(s) = \mathbb{E}_\pi\left[Q^\pi(s,a)\right]$ and $Q^\pi(s,a) = \hat{R}(s) + \gamma \mathbb{E}_{\mathcal{P}}\left[V^\pi(s')\right]$, corresponding to the return-to-go from state s (and action a). Given a reward model $\hat{R}(\cdot)$, we can say $\hat{R}(s^*), s^* \in \mathcal{B}^*$, indicates reward for successful states and $\hat{R}(s), s \in \mathcal{B}$, for all other states. Assuming that s^* transitions to itself, then for policy evaluation with mean-squared error (MSE), we can write the temporal difference (TD) targets for non-successful and successful states, $y : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, of Q-updates as

$$y(s, s') = \hat{R}(s) + \gamma \mathbb{E}_\pi\left[Q(s', a')\right], \quad (1) \qquad y(s^*, s^*) = \hat{R}(s^*) + \gamma \mathbb{E}_\pi\left[Q(s^*, a')\right]. \quad (2)$$

Learning a Multitask Agent from Examples. We alleviate the challenging exploration problem of EBC by introducing auxiliary control from examples (ACE), an application of the scheduled auxiliary control framework [2, 15]. A task $\mathcal{T}$ is defined by a task-specific example buffer $\mathcal{B}^*_{\mathcal{T}}$. EBC methods aim to exclusively complete the main task $\mathcal{T}_{\text{main}}$, while ACE adds auxiliary tasks $\mathcal{T}_{\text{aux}} = \{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$ during learning. We refer the set of all tasks as $\mathcal{T}_{\text{all}} = \mathcal{T}_{\text{aux}} \cup \{\mathcal{T}_{\text{main}}\}$. ACE agents have intentions and a scheduler, described next.

Intentions. For each task $\mathcal{T} \in \mathcal{T}_{\text{all}}$, the corresponding intention consists of a task-specific policy $\pi_{\mathcal{T}}$, Q-function $Q_{\mathcal{T}}$, and state-conditioned reward model $\hat{R}_{\mathcal{T}}$. ACE optimizes the task-specific policies by maximizing the policy optimization objective $\mathcal{L}(\pi; \mathcal{T}) = \mathbb{E}_{\mathcal{B}, \pi_{\mathcal{T}}}\left[Q_{\mathcal{T}}(s,a)\right]$. The task-specific Q-functions are optimized via minimization of the Bellman residual

$$\mathcal{L}(Q; \mathcal{T}) = \mathbb{E}_{\mathcal{B}, \pi_{\mathcal{T}}}\left[(Q_{\mathcal{T}}(s,a) - y_{\mathcal{T}}(s,s'))^2\right] + \mathbb{E}_{\mathcal{B}^*_{\mathcal{T}}, \pi_{\mathcal{T}}}\left[(Q_{\mathcal{T}}(s^*,a) - y_{\mathcal{T}}(s^*,s^*))^2\right], \quad (3)$$

where $y_{\mathcal{T}}$ are Eqs. (1) and (2) with task-specific reward $\hat{R}_{\mathcal{T}}$. Intuitively, each $\pi_{\mathcal{T}}$ aims to maximize the estimated task-specific value $Q_{\mathcal{T}}$.

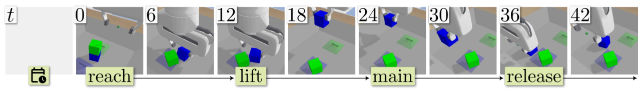


Figure 2: An example of fixed-period scheduler choices throughout an `Unstack-Stack` exploratory episode.

Scheduler. A scheduler periodically selects a policy $\pi_{\mathcal{T}}$ to execute within an episode (see Fig. 2). We use a weighted random scheduler (WRS) with hyperparameter $p_{\mathcal{T}_{\text{main}}}$ where the probability of choosing the main task or an auxiliary task is $p_{\mathcal{T}_{\text{main}}}$ and $p_{\mathcal{T}_k} = (1 - p_{\mathcal{T}_{\text{main}}})/K$, respectively. Following [15], we combine a WRS with a small set of simple handcrafted high-level trajectories (e.g., *reach* then *grasp* then *lift*). At test time, the scheduler is unused, and only π_{main} is evaluated.

Value Penalization in Example-Based Control. A scheduled multitask agent exhibits far more diverse behaviour than a single-task agent [2, 15]. We show in Figs. 5a and 5b that the buffer gener-

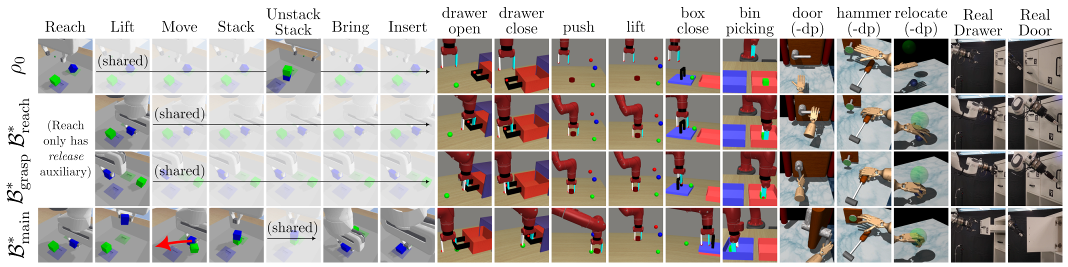


Figure 3: Samples from initial-state distribution ρ_0, auxiliary task examples $\mathcal{B}^*_{\text{aux}}$, and main task examples $\mathcal{B}^*_{\text{main}}$ for all tasks. The simulated Panda tasks additionally share $\mathcal{B}^*_{\text{release}}$ and $\mathcal{B}^*_{\text{lift}}$.

ated by this behavior, consisting of transitions resulting from multiple policies, can result in highly overestimated Q-values in EBC. This overestimation leads the policy to maximize an incorrect objective. Notice that regressing to TD targets Eqs. (1) and (2) will eventually satisfy the Bellman equation, but in the short term the TD targets do not satisfy $y(s, s') \leq y(s^*, s^*)$. We resolve this issue with a penalty for our TD updates for $s \in \mathcal{B}$. We add a minimum and a maximum $Q^\pi(s, a)$ as $Q^\pi_{\text{min}} = \hat{R}_{\text{min}}/(1-\gamma)$ and $Q^\pi_{\text{max}} = \mathbb{E}_{\mathcal{B}^*}\left[V^\pi(s^*)\right]$, where $\hat{R}_{\text{min}} \leq \hat{R}(s)$ for all $s \in \mathcal{B}$. Then, the value penalty is defined to be

$$\mathcal{L}^\pi_{\text{pen}}(Q) = \lambda \mathbb{E}_{\mathcal{B}}\big[\left(\max(Q(s,a) - Q^\pi_{\text{max}}, 0)\right)^2 + \left(\max(Q^\pi_{\text{min}} - Q(s,a), 0)\right)^2\big], \tag{4}$$

where $\lambda \geq 0$ is a hyperparameter. When $\lambda \to \infty$, Eq. (4) becomes a hard constraint. It immediately follows that $y(s, s') \leq y(s^*, s^*)$ holds with TD updates Eqs. (1) and (2). We add value penalization $\mathcal{L}^\pi_{\text{pen}}(Q)$ to the MSE loss as a regularization term for learning the Q-function.

3 Experiments

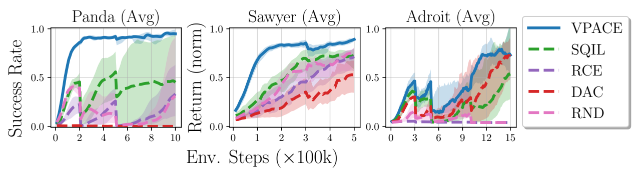
We aim to answer the following questions through our experiments: **(RQ1)** How does the sample efficiency of VPACE compare to EBC, inverse RL, and exploration-bonus baselines? **(RQ2)** How important is it to include value penalization (VP) with ACE? **(RQ3)** How does VPACE compare to algorithms that use full trajectories and true sparse rewards?

Baselines. We consider many baselines, including an approach based on recursively classifying examples (**RCE**, [1]), a learned reward model for off-policy RL (**DAC**, [4]), a defined reward model for off-policy RL (**SQIL**, [5]), and a combination of the best performing baseline (SQIL) with a method that provides an exploration bonus to unseen data (**RND**, [23, 24]). **VPACE** refers to SQIL with both VP and ACE applied, while **ACE** refers to SQIL with ACE and *without* VP applied.

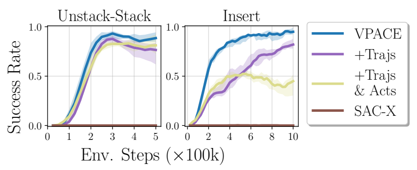
Environments. We conduct experiments in a large variety of tasks and environments, including those originally used in LfGP [15] and RCE [1]. Specifically, the tasks in [15] involve a simulated Franka Emika Panda arm, a blue and green block, a fixed "bring" area for each block, and a small slot with <1 mm tolerance for inserting each block. This environment provides various manipulation tasks that share the same state-action space. The tasks in [1] are a modified subset of those from [25], involving a simulated Sawyer arm, and three of the Adroit hand tasks originally presented in [26]. We also generate three modified delta-position (dp) Adroit hand environments. Finally, we study drawer and door opening tasks with a real Franka Emika Panda.

Implementation Details. All simulated tasks use 200 examples per task, while the real world tasks use 50 examples per task. All implementations are built on LfGP [15, 28] and SAC [29]. For more environment and implementation details, see appendix or our open-source code.

Main Task VP and ACE Benefits. To answer RQ1, we compare VPACE with existing EBC, inverse RL, and exploration-bonus approaches on all tasks and evaluate their success rates. Fig. 4a shows that VPACE has significant improvement over other approaches in both sample efficiency and final performance. We can observe that VPACE can consistently solve all tasks, while other EBC methods have significantly wider confidence intervals, especially in the Panda environment.

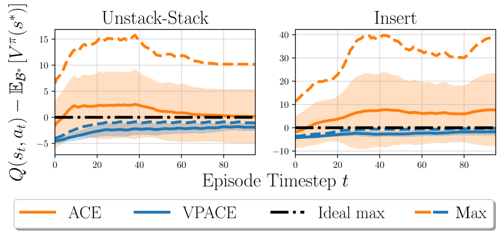


(a) Performance across simulated main tasks.

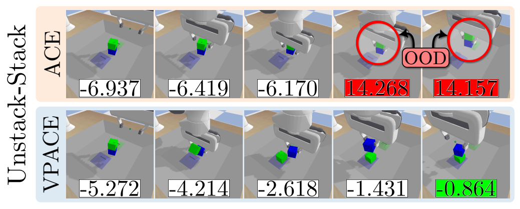


(b) Changes in the form of feedback.

Figure 4: Average across-task results separated by simulated environment, as well as feedback change results. Performance is an interquartile mean (IQM) across 5 evaluation timesteps and 5 seeds with shaded regions showing 95% stratified bootstrap confidence intervals [27].



(a) ACE without VP overestimates Q.



(b) $Q(s_t, a_t) - \mathbb{E}_{\mathcal{B}^*}[V^\pi(s^*)]$ for parts of episodes used to generate the results from Fig. 5a.

Figure 5: In `Unstack-Stack` and `Insert`, VP resolves Q overestimation.

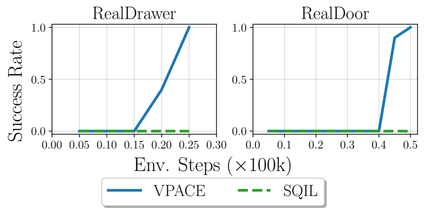


Figure 6: Real robot results.

We furthermore tested VPACE on real-life tasks (two right columns of Fig. 3), with results shown in Fig. 6. `RealDrawer` is learned by VPACE in about 100 minutes, while `RealDoor` is learned in about 200 minutes. The real world examples of success, for both main tasks and auxiliary tasks, are collected in less than a minute.

Q-Value Overestimation and Value Penalization. To answer RQ2 and verify that VP enforces $y(s, s') \leq y(s^*, s^*)$, we took snapshots of each learned `Unstack-Stack` and `Insert` agent, for both VPACE and ACE, at 300k steps and ran each policy for a single episode, recording per-timestep Q-values. Instead of showing Q-values directly, in Figs. 5a and 5b, we show $Q(s_t, a_t) - \mathbb{E}_{s^* \sim \mathcal{B}^*}[V(s^*)]$, which should be at most 0. ACE clearly violates $y(s, s') \leq y(s^*, s^*)$, while VPACE does not. Furthermore, Fig. 5b shows examples of the consequences of overestimated Q-values: for out-of-distribution (OOD) (s_t, a_t) pairs, $Q(s_t, a_t) - \mathbb{E}_{s^* \sim \mathcal{B}^*}[V(s^*)] > 0$, and the policy reaches these states instead of the true goal.

Comparison to Full Demonstrations and True Rewards. To answer RQ3, we test two variants of using full trajectories: a buffer containing both 200 examples and 200 (s, a) pairs from expert trajectories of states only (**+Trajs**), and the same for both states and actions (**+Trajs & Acts**), resulting in two variants of LfGP [15] with VP. For sparse rewards, we use the provided sparse-reward function from the task (**SAC-X**). Fig. 4b shows that the peak performance is *reduced* when learning with expert trajectories. We hypothesize that the divergence minimization objective leads to an effect, previously shown to occur in inverse RL [15], known as reward hacking [30]. Furthermore, the use of sparse rewards alone results in no success at all.

4 Conclusion

In this work, we presented VPACE—value-penalized auxiliary control from examples, where we coupled scheduled auxiliary control with value penalization in the example-based setting to significantly improve learning efficiency. Opportunities for future work include the further investigation of learned approaches to scheduling, as well as autonomously generating auxiliary task definitions.

Acknowledgments

We gratefully acknowledge the Digital Research Alliance of Canada and NVIDIA Inc., who provided the GPUs used in this work through their Resources for Research Groups Program and their Hardware Grant Program, respectively.

References

[1] B. Eysenbach, S. Levine, and R. Salakhutdinov. Replacing Rewards with Examples: Example-Based Policy Search via Recursive Classification. In *Advances in Neural Information Processing Systems (NeurIPS'21)*, Virtual, Dec. 2021.

[2] 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 *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, pages 4344–4353, Stockholm, Sweden, July 2018.

[3] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML'00)*, pages 663–670, July 2000. ISBN 1-55860-707-2. doi:10.2460/ajvr.67.2.323.

[4] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson. Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*, New Orleans, LA, USA, May 2019.

[5] S. Reddy, A. D. Dragan, and S. Levine. SQIL: Imitation Learning Via Reinforcement Learning with Sparse Rewards. In *International Conference on Learning Representations (ICLR'20)*, Apr. 2020.

[6] J. Ho and S. Ermon. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems (NIPS'16)*, Barcelona, Spain, Dec. 2016.

[7] J. Fu, K. Luo, and S. Levine. Learning Robust Rewards with Adverserial inverse Reinforcement Learning. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*, Vancouver, BC, Canada, Apr. 2018.

[8] J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine. Variational Inverse Control with Events: A General Framework for Data-Driven Reward Definition. In *Advances in Neural Information Processing Systems (NeurIPS'18)*, Montreal, Canada, Dec. 2018.

[9] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-End Robotic Reinforcement Learning without Reward Engineering. In *Robotics: Science and Systems (RSS'19)*, Freiburg im Breisgau, Germany, Apr. 2019.

[10] K. B. Hatch, B. Eysenbach, R. Rafailov, T. Yu, R. Salakhutdinov, S. Levine, and C. Finn. Contrastive Example-Based Control. In N. Matni, M. Morari, and G. J. Pappas, editors, *Learning for Dynamics and Control (L4DC'23)*, volume 211 of *Proceedings of Machine Learning Research*, pages 155–169, Philadelphia, PA, USA, June 2023. PMLR.

[11] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2):181–211, Aug. 1999. ISSN 00043702. doi:10.1016/S0004-3702(99)00052-1.

[12] A. Robert, C. Pike-Burke, and A. A. Faisal. Sample Complexity of Goal-Conditioned Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS'23)*, New Orleans, LA, USA, Dec. 2023.

[13] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine. Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning? In *Proceedings of the Neural Information Processing Systems (NeurIPS'19) Deep Reinforcement Learning Workshop*, Sept. 2019.

[14] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming Exploration in Reinforcement Learning with Demonstrations. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA'18)*, pages 6292–6299, Brisbane, Australia, May 2018. doi:10.1109/ICRA.2018.8463162.

[15] T. Ablett, B. Chan, and J. Kelly. Learning From Guided Play: Improving Exploration for Adversarial Imitation Learning With Simple Auxiliary Tasks. *IEEE Robotics and Automation Letters*, 8(3):1263–1270, Mar. 2023. ISSN 2377-3766. doi:10.1109/LRA.2023.3236882.

[16] T. Ablett, B. Chan, and J. Kelly. Learning from Guided Play: A Scheduled Hierarchical Approach for Improving Exploration in Adversarial Imitation Learning. In *Proceedings of the Neural Information Processing Systems (NeurIPS'21) Deep Reinforcement Learning Workshop*, Dec. 2021.

[17] G. Xiang, S. Li, F. Shuang, F. Gao, and X. Yuan. SC-AIRL: Share-Critic in Adversarial Inverse Reinforcement Learning for Long-Horizon Task. *IEEE Robotics and Automation Letters*, 9(4): 3179–3186, Apr. 2024. ISSN 2377-3766. doi:10.1109/LRA.2024.3366023.

[18] B. Wu, S. Nair, L. Fei-Fei, and C. Finn. Example-Driven Model-Based Reinforcement Learning for Solving Long-Horizon Visuomotor Tasks. *arXiv:2109.10312 [cs]*, Sept. 2021.

[19] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems (NIPS'17)*, Long Beach, CA, USA, Dec. 2017.

[20] J. Adamczyk, V. Makarenko, S. Tiomkin, and R. V. Kulkarni. Boosting Soft Q-Learning by Bounding. *Reinforcement Learning Journal*, 1(1), 2024.

[21] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative Q-Learning for Offline Reinforcement Learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, editors, *Advances in Neural Information Processing Systems (Neurips'20)*, Dec. 2020.

[22] S. James, K. Wada, T. Laidlow, and A. J. Davison. Coarse-to-Fine Q-Attention: Efficient Learning for Visual Robotic Manipulation Via Discretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13739–13748, 2022.

[23] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by Random Network Distillation. In *International Conference on Learning Representations (ICLR'19)*, New Orleans, LA, USA, May 2019. arXiv. doi:10.48550/arXiv.1810.12894.

[24] J. C. Balloch, R. Bhagat, G. Zollicoffer, R. Jia, J. Kim, and M. O. Riedl. Is Exploration All You Need? Effective Exploration Characteristics for Transfer in Reinforcement Learning, Apr. 2024. arXiv:2404.02235.

[25] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Conference on Robot Learning (CoRL'19)*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100, Osaka, Japan, Oct. 2019. PMLR.

[26] A. Rajeswaran*, V. Kumar*, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS'18)*, Pittsburgh, PA, USA, June 2018.

[27] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems (Neurips'21)*, volume 34, 2021.

[28] B. Chan. RL sandbox. https://github.com/chanb/rl_sandbox_public, 2020.

[29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, pages 1861–1870, Stockholm, Sweden, July 2018.

[30] J. Skalse, N. Howe, D. Krasheninnikov, and D. Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems (NeurIPS'22)*, New Orleans, LA, USA, Dec. 2022.

[31] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, June 2014. ISSN 00313203. doi:10.1016/j.patcog.2014.01.005.

[32] Y. Lin, A. S. Wang, G. Sutanto, A. Rai, and F. Meier. Polymetis. https://facebookresearch.github.io/fairo/polymetis/, 2021.

[33] S. K. S. Ghasemipour, R. S. Zemel, and S. S. Gu. A Divergence Minimization Perspective on Imitation Learning Methods. In *Conference on Robot Learning (CoRL'19)*, 2019.

[34] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev. Time Limits in Reinforcement Learning. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4042–4051. PMLR, 2018.

[35] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[36] S. Sinha, A. Mandlekar, and A. Garg. S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning. In *Conference on Robot Learning (CoRL'21)*, London, UK, Nov. 2021.

[37] X. Chen, C. Wang, Z. Zhou, and K. Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. *arXiv:2101.05982 [cs]*, Mar. 2021.

[38] K. Hatch, T. Yu, R. Rafailov, and C. Finn. Example-Based Offline Reinforcement Learning without Rewards. In *Advances in Neural Information Processing Systems (NeurIPS'21) Offline Reinforcement Learning Workshop*, Dec. 2021.

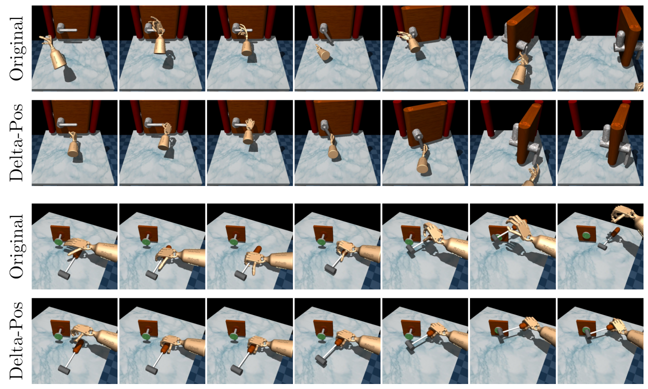


Figure 7: Learned VPACE-SQIL policies at the final training step for, from top to bottom, `door-human-v0`, `door-human-v0-dp`, `hammer-human-v0`, and `hammer-human-v0-dp`. Although the original versions of the environments are solved, the absolute position action space allows policies to execute very coarse actions that exploit the simulator (above, hitting the handle without grasping it and throwing the hammer, respectively), and would almost certainly cause damage to a real environment.

A Additional Environment, Algorithm, and Implementation Details

The following sections contain further details of the environments, tasks, auxiliary tasks, algorithms, and implementations used in our experiments.

A.1 Additional Environment Details

Compared with the original Panda tasks from LfGP [15], we switch from 20Hz to 5Hz control (finding major improvements in performance for doing so), improve handling of rotation symmetries in the observations, and remove the force-torque sensor since it turned out to have errors at low magnitudes. Crucially, these modifications did not require training new expert policies, since the same final observation states from the full trajectory expert data from [15] remained valid. Compared with the original LfGP tasks, we also remove `Move-block` as an auxiliary task from `Stack`, `Unstack-Stack`, `Bring` and `Insert`, since we found a slight performance improvement for doing so, and add `Reach`, `Lift`, and `Move-block` as main tasks. The environment was otherwise identical to how it was implemented in LfGP, including allowing randomization of the block and end-effector positions anywhere above the tray, using delta-position actions, and using end-effector pose, end-effector velocity, object pose, object velocity, and relative positions in the observations. For even further details of the original environment, see [15].

Since the Sawyer tasks from [1, 25] only contain end-effector position and object position by default, they do not follow the Markov property. To mitigate this, we train all algorithms in the Sawyer tasks with frame-stacking of 3 and add in gripper position to the observations, since we found that this, at best, improved performance for all algorithms, and at worst, kept performance the same.

A.2 Delta-Position Adroit Hand Environments

The Adroit hand tasks from [26] use absolute positions for actions. This choice allows even very coarse policies, with actions that would be unlikely to be successful in the real world, to learn to complete `door-human-v0` and `hammer-human-v0`, and also makes the intricate exploration required to solve `relocate-human-v0` very difficult. Specifically, VPACE-SQIL and several other baselines achieve high return in `door-human-v0` and `hammer-human-v0`, but the learned policies use unrealistic actions that exploit simulator bugs. As well, no methods are able to achieve any return in `relocate-human-v0` in 1.5M environment steps.

In the interest of solving `relocate-human-v0` and learning more skillful policies, we generated modified versions of these environments with delta-position action spaces. Furthermore, in

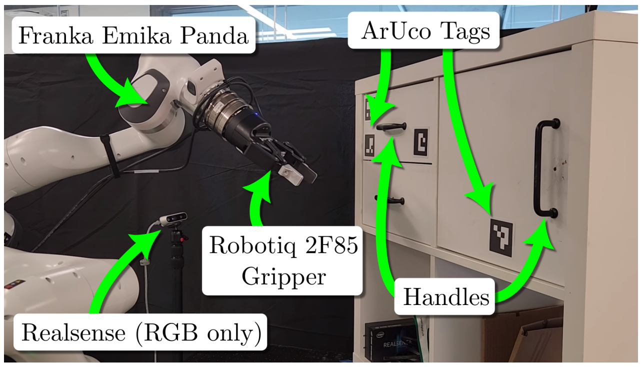


Figure 8: Experimental setup for our real environment and our `RealDrawer` and `RealDoor` tasks. The robot has a force-torque sensor attached, but it is not used for our experiments.

`relocate-human-v0-najp-dp`, the action space rotation frame was changed to be in the palm, rather than at the elbow, and, since relative positions between the palm, ball, and relocate goal are included as part of the state, we removed the joint positions from the state. In our experiments, these modified environments were called `door-human-v0-dp`, `hammer-human-v0-dp`, and `relocate-human-v0-najp-dp`. See Fig. 7 for a comparison of learned policies in each version of these environments.

A.3 Real World Environment Details

Fig. 8 shows our experimental platform and setup for our two real world tasks. In both `RealDrawer` and `RealDoor`, the observation space contains the end-effector position, an ArUco tag [31] to provide drawer or door position (in the frame of the RGB camera; we do not perform extrinsic calibration between the robot and the camera), and the gripper finger position. Observations also include two stacked frames, in lieu of including velocity, to better follow the Markov property. The action space in both contains delta-positions and a binary gripper command for opening and closing. The action space for `RealDrawer` is one-dimensional (allowing motion in a line), while the action space for `RealDoor` is two-dimensional (allowing motion in a plane). The initial state distribution for `RealDrawer` allows for initializing the end-effector anywhere within a 10 cm line approximately 25 cm away from the drawer handle when closed. For `RealDoor`, the initial state distribution is a 20 cm $\times$ 20 cm square, approximately 20cm away from the door handle when closed. Actions are supplied at 5 Hz.

For both environments, for evaluation only, success is determined by whether the drawer or door is fully opened, as detected by the absolute position of the ArUco tag in the frame of the RGB camera. Our robot environment code is built on Polymetis [32], and uses the default hybrid impedance controller that comes with the library. To reduce environmental damage from excessive forces and torques, we reduced Cartesian translational stiffness in all dimensions from 750 N/m to 250 N/m, and the force and torque limits in all dimensions from 40 N and 40 Nm to 20 N and 20 Nm.

A.4 Additional Task Details

Success examples for the Panda environments were gathered by taking s_T from the existing datasets provided by [15]. Success examples for main tasks from the Sawyer environments were generated using the same code from [1], in which the success examples were generated manually given knowledge of the task. Auxiliary task data was generated with a similar approach. Success examples for the Adroit hand environments were generated from the original human datasets provided by [26]. Success examples for our real world tasks were generated by manually moving the robot to a small set of successful positions for each auxiliary task and main task. All Panda main tasks use the the auxiliary tasks *release*, *reach*, *grasp*, and *lift*.

There are two specific nuances that were left out of the main text for clarity and brevity: (i) the `Reach` main task only uses *release* as an auxiliary task (since it also acts as a "coarse" reach), and (ii) half of the *release* dataset for each task is specific to that task (e.g., containing insert or stack data), as was the case in the original datasets from [15]. For the Sawyer, Hand, and real Panda environments, because the observation spaces are not shared, each task has its own separate *reach* and *grasp* data.

A.5 Additional Algorithm Details

Algorithm 1 Value-Penalized Auxiliary Control from Examples (VPACE)

Input: Example state buffers $\mathcal{B}^*_{\text{main}}, \mathcal{B}^*_1, \ldots, \mathcal{B}^*_K$, scheduler period ξ, sample batch size N and N^*, and discount factor γ

Parameters: Intentions $\pi_{\mathcal{T}}$ with corresponding Q-functions $Q_{\mathcal{T}}$ (and optionally discriminators $D_{\mathcal{T}}$), and scheduler π_S (e.g. with Q-table Q_S)

```
1:  Initialize replay buffer $\mathcal{B}$
2:  for $t = 1, \ldots,$ do
3:      # Interact with environment
4:      For every $\xi$ steps, select intention $\pi_{\mathcal{T}}$ using $\pi_S$
5:      Select action $a_t$ using $\pi_{\mathcal{T}}$
6:      Execute action $a_t$ and observe next state $s'_t$
7:      Store transition $\langle s_t, a_t, s'_t \rangle$ in $\mathcal{B}$
8:
9:      # Optionally update discriminator $D_{\mathcal{T}'}$ for each task $\mathcal{T}'$
10:     Sample $\{s_i\}_{i=1}^{N} \sim \mathcal{B}$
11:     for each task $\mathcal{T}'$ do
12:         Sample $\{s_i^*\}_{i=1}^{N^*} \sim \mathcal{B}^*_k$
13:         Update $D_{\mathcal{T}'}$ using GAN + Gradient Penalty
14:     end for
15:
16:     # Update intentions $\pi_{\mathcal{T}'}$ and Q-functions $Q_{\mathcal{T}'}$ for each task $\mathcal{T}'$
17:     Sample $\{(s_i, a_i)\}_{i=1}^{N} \sim \mathcal{B}$
18:     for each task $\mathcal{T}'$ do
19:         Sample $\{s_i^*\}_{i=1}^{N^*} \sim \mathcal{B}^*_{\mathcal{T}'}$
20:         Sample $a_i^* \sim \pi_{\mathcal{T}'}(s_i^*)$ for $i = 1, \ldots, N^*$
21:         Compute rewards $\hat{R}_{\mathcal{T}'}(s_i)$ and $\hat{R}_{\mathcal{T}'}(s_j^*)$ for $i = 1, \ldots, N$ and $j = 1, \ldots, N^*$
22:         # Compute value penalization terms, see Appendix A.6.1
23:         Compute $Q_{\text{max}}^{\pi_{\mathcal{T}'}} = \frac{1}{N^*} \sum_{j=1}^{N^*} Q(s_j^*, a_j^*)$
24:         Compute $Q_{\text{min}}^{\pi_{\mathcal{T}'}} = \min(\wedge_{i=1}^{N} \hat{R}_{\mathcal{T}'}(s_i), \wedge_{j=1}^{N^*} \hat{R}_{\mathcal{T}'}(s_j^*))/(1-\gamma)$
25:     end for
26:     Update $\pi$ following Eq. (5)
27:     Update $Q$ following Eq. (3) with value penalization Eq. (4)
28:
29:     # *Optional* Update learned scheduler $\pi_S$
30:     if at the end of effective horizon then
31:         Compute main task return $G_{\mathcal{T}_{\text{main}}}$ using reward estimate from $D_{\text{main}}$
32:         Update $\pi_S$ (e.g. update Q-table $Q_S$ using EMA and recompute Boltzmann distribution)
33:     end if
34: end for
```

Algorithm 1 shows a summary of VPACE, built on LfGP [15] and SAC-X [2]. As a reminder, our multi-policy objective function is

$$\mathcal{L}(\pi; \mathcal{T}) = \mathbb{E}_{\mathcal{B}, \pi_{\mathcal{T}}} \left[Q_{\mathcal{T}}(s, a) \right], \tag{5}$$

and our minimum and and maximum Q values are

$$Q_{\text{min}}^{\pi} = \hat{R}_{\text{min}}/(1-\gamma), \tag{6}$$

Algorithm	Value Penalization	Sched. Aux. Tasks	Reward Model	TD Error Loss	Source
VPACE	✓	✓	SQIL	MSE	Ours
ACE	✗	✓	SQIL	MSE	Ours
VPACE-DAC	✓	✓	DAC	MSE	Ours
ACE-RCE	✗	✓	RCE	BCE	Ours
VP-SQIL	✓	✗	SQIL	MSE	Ours
DAC	✗	✗	DAC	MSE	Ours
SQIL	✗	✗	SQIL	MSE	Ours
RCE	✗	✗	RCE	BCE	Ours
SQIL+RND	✗	✗	SQIL	MSE	Ours
RCE (theirs)	✗	✗	RCE	BCE	[1]
SQIL-BCE	✗	✗	SQIL	BCE	[1]

Table 1: Major differences between algorithms studied in this work. MSE refers to mean squared error, while BCE refers to binary cross entropy.

and

$$Q^{\pi}_{\text{max}} = \mathbb{E}_{\mathcal{B}^*}\left[V^{\pi}(s^*)\right]. \tag{7}$$

Table 1 shows a breakdown of some of the major differences between all of the algorithms studied in this work.

A.6 Additional Implementation Details

In this section, we list some specific implementation details of our algorithms. We only list parameters or choices that may be considered unique to this work, but a full list of all parameter choices can be found in our code. We also provide the VPACE pseudocode in Algorithm 1, with blue text only applying to learned discriminator-based reward functions.

Whenever possible, all algorithms and baselines use the same modifications. Table 2 also shows our choices for common off-policy RL hyperparameters as well as choices for those introduced by this work.

DAC reward function: for VPACE-DAC and VP-DAC, although there are many options for reward functions that map D to $\hat{R}$ [33], following [15, 4, 7], we set the reward to $\hat{R}_{\mathcal{T}}(s) = \log\left(D_{\mathcal{T}}(s)\right) - \log\left(1 - D_{\mathcal{T}}(s)\right)$.

n-step returns and entropy in TD error: following results from [1], we also add n-step returns and remove the entropy bonus in the calculation of the TD error for all algorithms in all Sawyer and Adroit environments, finding a significant performance gain for doing so.

Absorbing states and terminal states: for all algorithms, we do not include absorbing states (introduced in [4]) or terminal markers (sometimes referred to as "done"), since we found that both of these additions cause major bootstrapping problems when environments only terminate on timeouts, and timeouts do not necessarily indicate failure. Previous work supports bootstrapping on terminal states when they are caused by non-failure timeouts [34].

SQIL labels for policy data: The original implementation of SQIL uses labels of 0 and 1 for TD updates. We found that changing the label from 0 to -1 improved performance.

Reward Scaling of .1: we use a reward scaling parameter of .1 for all implementations. Coupled with a discount rate $\gamma = 0.99$ (common for much work in RL), this sets the expected minimum and maximum Q values for SQIL to $\frac{-.1}{1-\gamma} = -10$ and $\frac{.1}{1-\gamma} = 10$.

No multitask weight sharing: intuitively, one may expect weight sharing to be helpful for multitask implementations. We found that it substantially hurt performance, so all of our multitask methods do not share weights between tasks or between actor and critic. However, the multitask discriminator in VPACE-DAC *does* have an initial set of shared weights due to its significantly poorer performance without this choice.

Table 2: Hyperparameters shared between all algorithms, unless otherwise noted.

General	
Total Interactions	Task-specific (see Appendix B)
Buffer Size	Same as total interactions
Buffer Warmup	5k
Initial Exploration	10k
Evaluations per task	50 (Panda), 30 (Sawyer/Adroit)
Evaluation frequency	25k (Panda), 10k (Sawyer/Adroit)
Learning	
γ	0.99
$\mathcal{B}$ Batch Size	128
$\mathcal{B}^*$ Batch Size	128
Q Update Freq.	1 (sim), 4 (real)
Target Q Update Freq.	1 (sim), 4 (real)
π Update Freq.	1 (sim), 4 (real)
Polyak Averaging	1e-3
Q Learning Rate	3e-4
π Learning Rate	3e-4
D Learning Rate	3e-4
α Learning Rate	3e-4
Initial α	1e-2
Target Entropy	$-\text{dim}(a)$
Max. Gradient Norm	10
Weight Decay (π, Q, D)	1e-2
D Gradient Penalty	10
Reward Scaling	0.1
SQIL Labels (Appendix A.6	$(-1, 1)$
Expert Aug. Factor (Appendix A.6.3)	0.1
Value Penalization (VP)	
λ	10
Q^{π}_{max}, Q^{π}_{min} num. filter points (Appendix A.6.1)	50
Auxiliary Control (ACE) Scheduler (Appendix A.6.2)	
Num. Periods	8 (Panda), 5 (Sawyer/Adroit)
Main Task Rate	0.5 (Panda), 0.0 (Sawyer/Adroit)
Handcraft Rate	0.5 (Panda), 1.0 (Sawyer/Adroit)

$\mathcal{B}^*$ sampling for Q: in SQIL, DAC and RCE, we sample from both $\mathcal{B}$ and $\mathcal{B}^*$ for Q updates, but not for π updates (which only samples from $\mathcal{B}$). The original DAC implementation in [4] only samples $\mathcal{B}^*$ for updating D, sampling only from $\mathcal{B}$ for updating Q.

All other architecture details, including neural network parameters, are the same as [15], which our own implementations are built on top of. Our code is built on top of the code from [15], which was originally built using [28].

A.6.1 Maintaining Q^{π}_{max}, Q^{π}_{min} Estimates for Value Penalization

Our approach to value penalization requires maintaining estimates for or choosing Q^{π}_{max} and Q^{π}_{min}. In both DAC and SQIL, the estimate of Q^{π}_{max} comes from taking the mini-batch of data from $\mathcal{B}^*$, passing it through the Q function, taking the mean, and then using a median moving average filter to maintain an estimate. The "Q^{π}_{max}, Q^{π}_{min} num. filter points" value from Table 2 refers to the size of this filter. We chose 50 and used it for all of our experiments. We set Q^{π}_{min} to

$$Q^{\pi}_{\text{min}} = \frac{\text{rew. scale} \times \min(\hat{R}(s))}{1-\gamma}, \tag{8}$$

where in SQIL, $\min(\hat{R}(s)) = \hat{R}(s)$ is set to 0 or -1, and in DAC, we maintain an estimate of the minimum learned reward $\min(\hat{R})$ using a median moving average filter with the same length as the one used for Q^{π}_{max} .

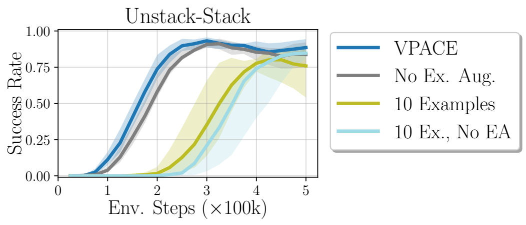


Figure 9: Variations of example augmentation. Our example augmentation scheme provides a small but noticeable bump in performance, which is magnified with when fewer expert examples are used.

A.6.2 Scheduler Choices

Our *ACE* algorithms use the same approach to scheduling from [15]. Specifically, we use a weighted random scheduler (WRS) combined a small set of handcrafted high-level trajectories. The WRS forms a prior categorical distribution over the set of tasks, with a higher probability mass $p_{\mathcal{T}_{\text{main}}}$ (Main Task Rate in Table 2) for the main task and $\frac{p_{\mathcal{T}_{\text{main}}}}{K}$ for all other tasks. Additionally, we choose whether to uniformly sample from a small set of handcrafted high-level trajectories, instead of from the WRS, at the *Handcraft Rate* from Table 2.

Our selections for handcrafted trajectories are quite simple, and reusable between main tasks within each environment. In the Panda tasks, there are eight scheduler periods per episode and four auxiliary tasks (*reach*, *grasp*, *lift*, *release*), and the handcrafted trajectory options are:

1. *reach*, *lift*, *main*, *release*, *reach*, *lift*, *main*, *release*
2. *lift*, *main*, *release*, *lift*, *main*, *release*, *lift*, *main*
3. *main*, *release*, *main*, *release*, *main*, *release*, *main*, *release*

In the Sawyer and Adroit environments, we actually found that the WRS was unnecessary to efficiently learn the main task, and simply used two handcrafted high-level trajectories. In these environments, there are five scheduler periods per episode and two auxiliary tasks (*reach*, *grasp*), and the handcrafted trajectory options are:

1. *reach*, *grasp*, *main*, *main*, *main*
2. *main*, *main*, *main*, *main*, *main*

A.6.3 Expert Data Augmentation

We added a method for augmenting our expert data to artificially increase dataset size. The approach is similar to other approaches that simply add Gaussian or uniform noise to data in the buffer [35, 36]. In our case, we go one step further than the approach from [36], and first calculate the per-dimension standard deviation of each observation in $\mathcal{B}^*$, scaling the Gaussian noise added to each dimension of each example based on the dimension's standard deviation. For example, if a dimension in $\mathcal{B}^*$ has zero standard deviation (e.g., in `Insert`, the pose of the blue block is always the same), it will have no noise added by our augmentation approach. The parameter "Expert Aug. Factor" from Table 2 controls the magnitude of this noise, after our per-dimension normalization scheme.

In Fig. 9, we show the results of excluding expert augmentation, where there is a clear, if slight, performance decrease when it is excluded, which is even more pronounced with a smaller $\mathcal{B}^*_{\mathcal{T}}$ size. All methods and baselines from our own implementation use expert data augmentation.

A.6.4 Other Ablation Details

In our ablation experiments, we included two baselines with full trajectory data, in addition to success examples. We added 200 (s, a) pairs from full expert trajectories to make datasets comparable to the datasets from [15], where they used 800 expert (s, a) pairs, but their environment was run at 20Hz instead of 5Hz, meaning they needed four times more data to have roughly the same total number of expert trajectories. We generated these trajectories using high-performing policies from our main experiments, since the raw trajectory data from [15] would not apply given that we changed the control rate from 20Hz to 5Hz.

A.6.5 Real Panda Implementation Details

While most of the design choices in Appendix A.6 apply to all environments tested, our real Panda environment had some small specific differences, mostly due to the complications of running reinforcement learning in the real world. We list the differences here, but for an exhaustive list, our open source code contains further details.

Maximum episode length: The maximum episode length for both `RealDrawer` and `RealDoor` is 1000 steps, or 200 seconds in real time. This was selected to reduce how often the environment had to be reset, which is time consuming. Running episodes for this long, and executing actions at 5 Hz, our environments complete 5000 environment steps in roughly 20 minutes. The extra time is due to the time to reset the environment after 1000 steps or after a collision. VPACE took approximately 100 minutes to learn to complete `RealDrawer` consistently, and about 200 minutes to learn to complete the more difficult `RealDoor`.

Shorter initial exploration: To attempt to learn the tasks with fewer environment samples, we reduce buffer warmup to 500 steps, and initial random exploration to 1000 steps.

Frame stack: For training, we stacked two regular observations to avoid state aliasing.

Ending on success: We ended episodes early if they were determined to be successful at the main task only. Although this is not necessary for tasks to be learned (and this information was *not* provided to the learning algorithm), it gave us a way to evaluate training progress.

Extra gradient steps: To add efficiency during execution and training, we completed training steps during the gap in time between an action being executed and an observation being gathered. Instead of completing a single gradient step at this time, as is the case for standard SAC (and VPACE), we completed four gradient steps, finding in simulated tasks that this gave a benefit to learning efficiency without harming performance. Previous work [15], [37] has found that increasing this update rate can reduce performance, but we hypothesize that our value penalization scheme helps mitigate this issue.

Collisions: If the robot is detected to have exceeded force or torque limits (20 N and 20 Nm in our case, respectively), the final observation prior to collision is recorded, and the environment is immediately reset. There are likely more efficient ways to handle such behaviour, but we did not investigate any in this work.

B Additional Performance Results

In this section, we expand upon our performance results and our Q-value overestimation analysis.

B.1 Expanded Main Task Performance Results

Fig. 10 shows expanded results for our simulated environments, with each baseline shown for each individual environment.

B.2 ACE Reward Model Comparison

The SAC-X framework, which we describe as ACE when used with example states only, is agnostic to the choice of reward model. Therefore, we also completed experiments in each of our simulated environments with DAC and with RCE as the base reward model, instead of SQIL, which was used for all of our main VPACE results. The results are shown in Fig. 11, where it is clear that VPACE with SQIL learns more efficiently than VPACE with DAC and with higher final performance than

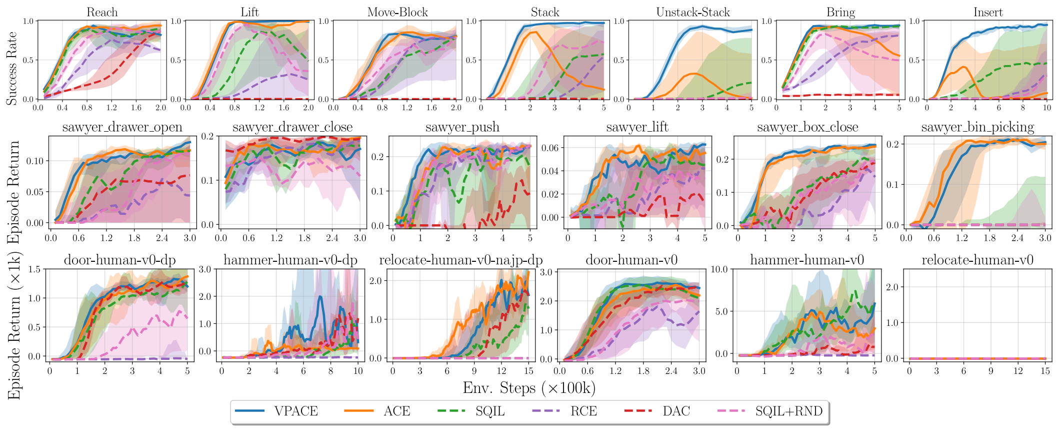


Figure 10: Sample efficiency performance plots for main task only for all simulated main tasks, including baselines not shown in the main text. Methods introduced in this work have solid lines, while baselines are shown with dashed lines. Performance is an interquartile mean (IQM) across 5 timesteps and 5 seeds with shaded regions showing 95% stratified bootstrap confidence intervals [27].

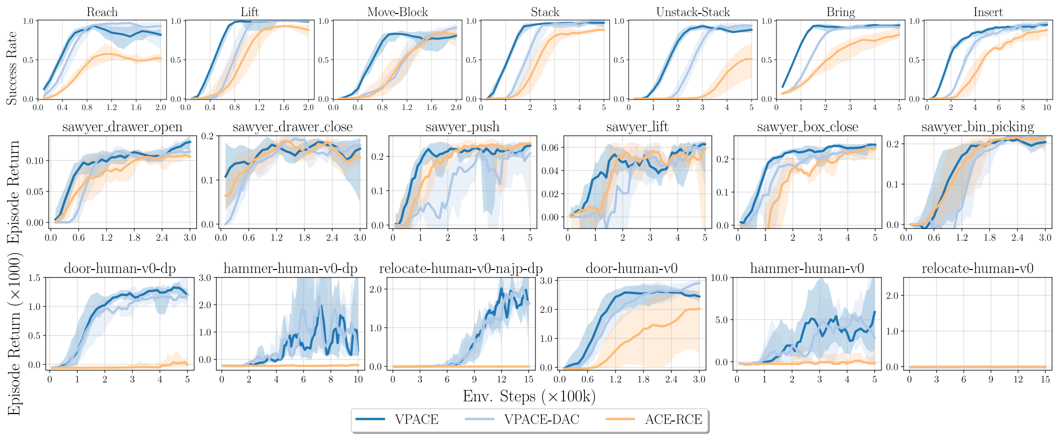


Figure 11: Comparison of underlying reward models when used with ACE. Our main results (VPACE) used SQIL as the reward model. VPACE-DAC eventually reaches similar performance to VPACE-SQIL, but tends to take longer, and ACE-RCE often has far poorer performance than both.

ACE with RCE. We do not use value penalization (VP) for RCE, since RCE uses a classification loss for training Q that would not benefit from the use of value penalization.

B.3 Single-task Reward Model Comparison

We compare SQIL, DAC, and RCE , without adding value penalization (VP) or auxiliary control from examples (ACE) in Fig. 14, with two additional, more recent EBC baselines: **EMBER** [18] and RCE combined with conservative Q-learning (**RCE-CQL**)[21], [38]). We find that both EMBER and RCE-CQL perform quite poorly, and much more poorly than SQIL, DAC, and RCE, further justifying their use as our primary reward models in our main experiments.

B.4 Single-task Value Penalization

Our scheme for value penalization, while initially motivated by the use of ACE, can be used in the single task regime. In Fig. 12, we compare the performance of VPACE, SQIL, and SQIL with value penalization, but *without* auxiliary tasks (VP-SQIL). VP-SQIL either has no effect on performance or results in an improvement over SQIL, as expected, but is still strongly outperformed by VPACE on the more complicated tasks, such as `Stack`, `Unstack-Stack`, `Insert`, `sawyer_box_close`, `sawyer_bin_picking`, and the `dp` variant of `relocate-human-v0`.

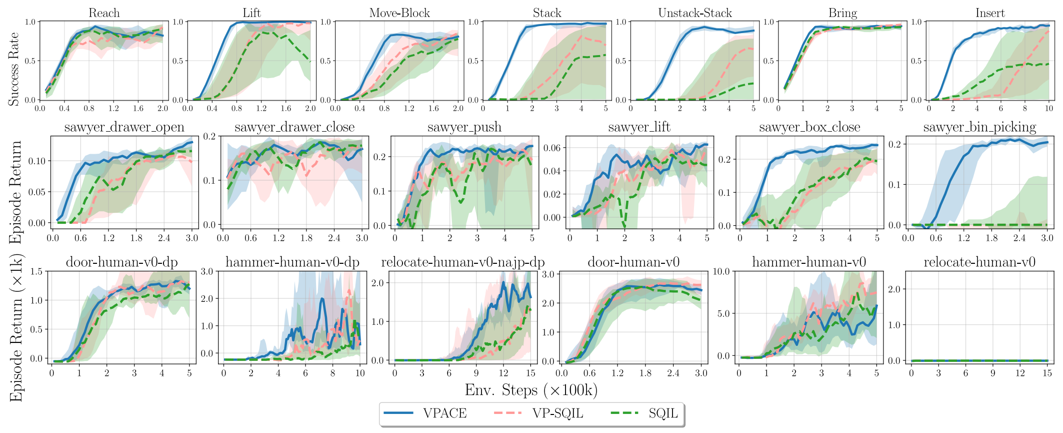


Figure 12: Comparison of different variants of SQIL: both VP and ACE (VPACE), VP-only (VP-SQIL), or SQIL alone. VP-SQIL generally either results in an improvement or has no effect on performance, compared with SQIL, but the exclusion of ACE results in far poorer performance than VPACE, especially for the most complex tasks.

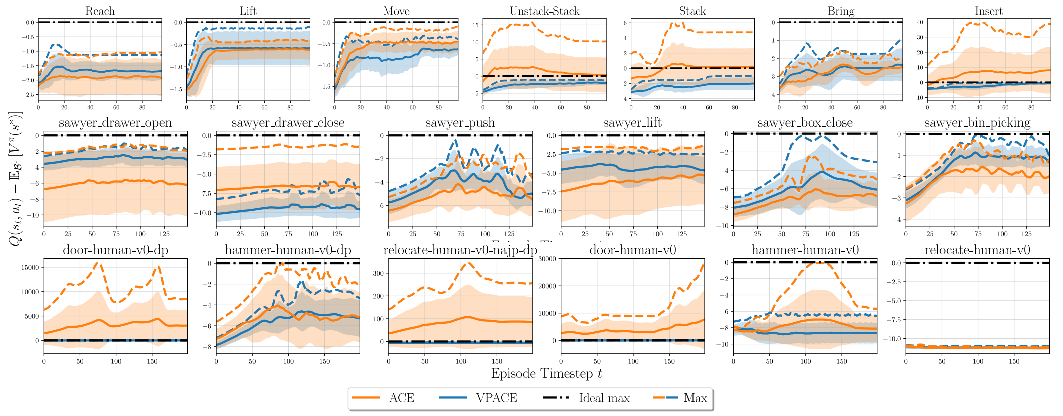


Figure 13: Comparison of different variants of SQIL: both VP and ACE (VPACE), VP-only (VP-SQIL), or SQIL alone. VP-SQIL generally either results in an improvement or has no effect on performance, compared with SQIL, but the exclusion of ACE results in far poorer performance than VPACE, especially for the most complex tasks.

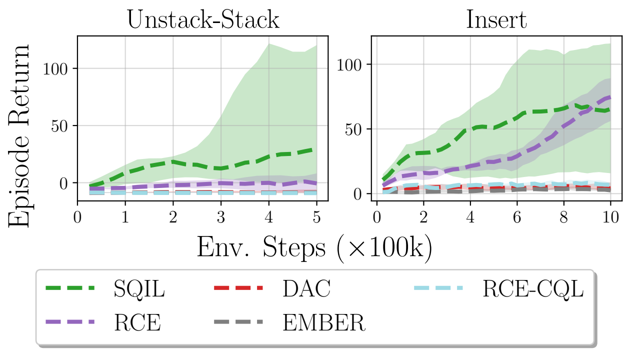


Figure 14: Results for different reward models for EBC, excluding the use of both value penalization (VP) and auxiliary control from examples (ACE).

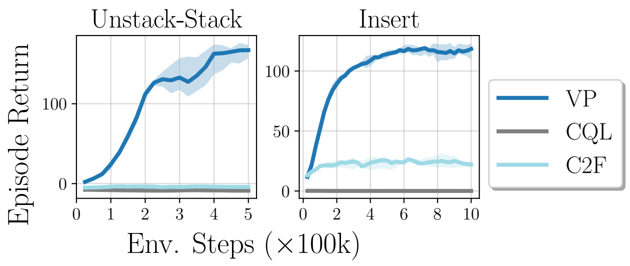


Figure 15: Results for different example-based approaches to regularizing Q estimates. We measure performance using return instead of success rate as the alternatives achieved no success at all.

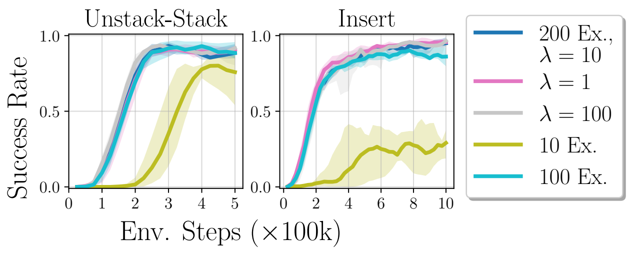


Figure 16: Variations of success example quantity and λ value.

B.5 Q-Value Overestimation and Penalization – All Environments

Fig. 13 shows the same value overestimation analysis plots shown in the original paper for all tasks in. The results for other difficult tasks also show clear violations of $y(s, a) \leq y(s^*, a^*)$, and tasks in which this rule is violated also often have poorer performance. Intriguingly, although the rule is severely violated for many Adroit hand tasks, ACE without VP still has reasonable performance in some cases. This shows that highly uncalibrated Q estimates can still, sometimes, lead to adequate performance. We hypothesize that this occurs because these tasks do not necessarily need $s_T \sim \mathcal{B}$ to match $s^* \sim \mathcal{B}^*_{\text{main}}$ to achieve high return, but we leave investigating this point to future work.

B.6 Alternative Q-Value Regularization Approaches

To understand the importance of the $y(s, s') \leq y(s^*, s^*)$ constraint, we investigate other choices of regularization techniques (RQ3), in particular a L2 regularizer on Q-values (**C2F**, [22]) and a regularizer that penalizes the Q-values of out-of-distribution actions (**CQL**, [21]). Fig. 15 indicates that using other existing regularization techniques does not enable ACE to perform well. We suspect that the L2 regularizer may be too harsh of a penalty on all learning, while [21] does not have any penalization on the magnitudes of Q-values in general, meaning that they can potentially still have uncontrollable bootstrapping error.

B.6.1 Expert Data Quantity and λ Value

Finally, to examine the robustness of our approach, we also compare various values for the quantity of examples of success (200 / task for main experiments) as well as the value of λ, which controls the strength of value penalization (10.0 for main experiments). Fig. 16 shows that changing the value of λ to 1 and 100 has no effect on performance. Dropping from 200 to 100 examples has only a minor negative impact on performance in `Insert`, while the drop to 10 examples has a significant negative effect. However, even with only 10 examples, performance on Unstack-Stack still far exceeds all other baselines.