Skill-Based Reinforcement Learning with Intrinsic Reward Matching

Abstract

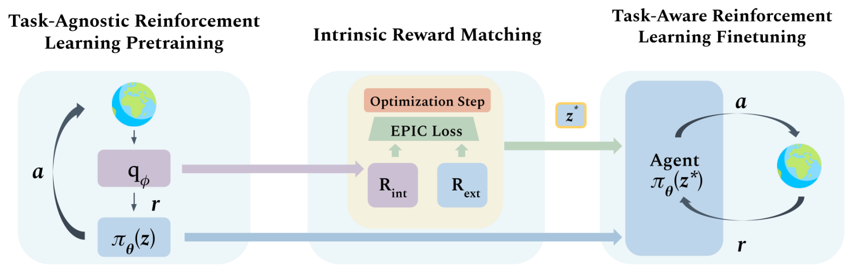
While unsupervised skill discovery has shown promise in autonomously acquiring behavioral primitives, there is still a large methodological disconnect between task-agnostic skill pretraining and downstream, task-aware finetuning. We present Intrinsic Reward Matching (IRM), which unifies these two phases of learning via the *skill discriminator*, a pretraining model component often discarded during finetuning. Conventional approaches finetune pretrained agents directly at the policy level, often relying on expensive environment rollouts to empirically determine the optimal skill. However, often the most concise yet complete description of a task is the reward function itself, and skill learning methods learn an *intrinsic* reward function via the discriminator that corresponds to the skill policy. We propose to leverage the skill discriminator to *match* the intrinsic and downstream task rewards and determine the optimal skill for an unseen task without environment samples on a Fetch tabletop manipulation task suite.

1 Introduction

Generalist agents must possess the ability to execute a diverse set of behaviors and flexibly adapt them to complete novel tasks. Recent progress in scaling multitask reinforcement learning [25, 17] has revealed the potential of multitask agents to encode vast skill repertoires, rivaling the performance of specialist agents and even generalizing to out-of-distribution tasks. Moreover, skill-based unsupervised RL [18, 21, 27] shows promise of acquiring similarly useful behaviors but without the expensive per-task supervision required for conventional multitask RL. However, such approaches lack a principled framework for connecting unsupervised pretraining and downstream finetuning. The current state-of-the-art leverages inefficient skill search methods at the policy level such as performing a sampling-based optimization or sweeping a coarse discretization of the skill space [19]. However, such methods still exhibit key limitations; namely they (1) rely on expensive environment trials to evaluate which skill is optimal, and (2) are likely to select suboptimal behaviors as the continuous skill space grows due to the curse of dimensionality.

In this work, we present Intrinsic Reward Matching (IRM), a scalable algorithmic methodology for unifying unsupervised skill pretraining and downstream task finetuning by leveraging the learned intrinsic reward function parameterized by the skill discriminator. Centrally, we introduce a novel approach to leveraging the intrinsic reward model as a multitask reward function that, via interaction-free task inference, enables us to select the optimal pretrained policy for the extrinsic task reward. During pretraining, unsupervised skill discovery methods learn a discriminator-parameterized, family of reward functions corresponding to a family of policies, or skills, through a shared latent code. Instead of discarding the discriminator during finetuning as done in prior work, we observe that the discriminator is an effective task specifier for its corresponding policy that can be *matched* with the extrinsic reward. Our approach views the extrinsic reward as a distribution with measurable proximity to a pretrained multitask reward distribution and formulates an optimization with respect to skills over a reward distance metric called EPIC [12].

Preprint. Under review.



Task-Agnostic Reinforcement Learning Pretraining
Intrinsic Reward Matching
Task-Aware Reinforcement Learning Finetuning
a
q_ϕ
r
$\pi_\theta(z)$
Optimization Step
EPIC Loss
R_{int}
R_{ext}
z^*
Agent
$\pi_\theta(z^*)$
a
r

Figure 1: Intrinsic Reward Matching (IRM) Framework. IRM takes place in the middle of three training stages: (1) Task-agnostic RL pretraining learns skill primitives in conjunction with a skill discriminator $q_\phi(\tau, z)$, described in Section 2. (2) With no environment interaction, IRM minimizes the EPIC Loss between the intrinsic reward parameterized by the discriminator and the extrinsic reward, which we assume access to, with respect to the skill vector z. (3) The skill policy conditioned on the optimal z^* finetunes to task reward to solve the downstream task.

2 Background

Unsupervised Skill Pretraining: The skill learning literature has long sought to design agents that autonomously acquire structured behaviors in new environments [30, 29, 24]. Recent work in competence-based unsupervised RL proposes generic objectives encouraging the discovery of skills representing diverse and useful behaviors [7, 27, 18]. A skill is defined as a latent code vector $z \in \mathcal{Z}$ that indexes the conditional policy $\pi(a|s, z)$. In order to learn such a policy, this class of skill pretraining algorithms maximizes the mutual information between sampled skills and their resulting trajectories τ [13, 6, 26] :

$$I(\tau; z) = \mathcal{H}(z) - \mathcal{H}(z|\tau) = \mathcal{H}(\tau) - \mathcal{H}(\tau|z) \tag{1}$$

Commonly, methods consider $\tau = s$ or $\tau = (s, s')$, and we leave more details in Appendix 5.8. Since the mutual information $I(s; z)$ is intractable to calculate in practice, competence-based methods instead maximize a variational lower bound proposed in [2] which is parameterized by a learned neural network function $q_\phi(\tau, z)$ called a skill discriminator. This discriminator, along with other terms independent of z, parameterizes an intrinsic reward that the skill policy $\pi(\cdot|s, z)$ maximizes during pretraining. Given an unseen task specification, the agent needs to infer which skill will finetune to solve the task with the fewest samples.

Pretrained Multitask Reward Functions: We observe that the intrinsic reward function learned during skill pretraining can be viewed as a multitask reward function, where the continuous skill code z determines the task. In other words, we have some function:

$$\mathcal{R}^{int}(\tau, z) := \text{VLB}(\tau, z) \tag{2}$$

where VLB $\leq I(\tau, z)$ is the variational lower bound proposed in [2] (τ is a trajectory, e.g. (s, s')). Since skill discovery algorithms maximize $I(\tau, z)$, we view its parameterized lower bound VLB as a multitask reward function: scoring transitions based on alignment with a skill code [18].

EPIC Reward Distance: We can formalize a general notion of reward function similarity by equivalent-policy invariant comparison (EPIC) as established in [12]. EPIC defines a distance metric, D_{EPIC}, between two reward functions such that similar reward functions induce similar optimal policies. We provide additional details in Appendix 5.1.

3 Intrinsic Reward Matching

A multitask reward function that can supervise the learning of diverse behaviors is useful in its own right. However, in the case of skill-based RL, we have additionally learned a corresponding $\pi(a|s, z)$. Therefore, for any “task" that can be specified by our intrinsic reward function, we already have an optimal policy, so long as we condition on the corresponding skill. If we have learned a sufficiently diverse library of skills, we might expect that some of our skills share behavioral similarity to the optimal policy for the downstream task. It thus also holds that the corresponding intrinsic reward for that skill is a semantically similar task specification to the downstream task.

Task	IRM CEM	IRM GD	IRM Rand	Env Roll.	Env CEM	GS	Rand
Fetch Reach	95.9 ± 1.0	87.5 ± 0.20	92.5 ± 1.1	85.0 ± 6.2	87.8 ± 1.9	**97.3** ± **0.00**	16.7 ± 19
Fetch Push	**80.2** ± **2.5**	73.1 ± 0.48	77.6 ± 2.7	74.3 ± 0.92	75.4 ± 2.6	72.1 ± 0.00	51.5 ± 12.5

Table 1: IRM with various optimization methods compared to environment rollout-based skill selection and random skill selection. IRM based methods rival or exceed skill selection baselines that are reliant on expensive environment trials.

Given this interpretation of intrinsic reward, we posit that the task of identifying which our pretrained skills to apply to a downstream task can be reframed as inferring which task in our multitask reward function is most similar to the downstream task. Moreover, we should hope to find the skill code z that produces the reward function most semantically aligned with the downstream task reward.

With this formalism, we can formulate the task inference problem as the following optimization:

$$z^* = \arg\min_z D_{\text{EPIC}}(R^{int}(\tau, z), R^{ext}(\tau)) \tag{3}$$

in order to find z^* most aligned with the task reward. Moreover, Equation 3 performs a minimization of a novel loss we name the *EPIC loss* with respect to the skill parameter z. By EPIC's equivalence class invariance [12], we know that if the EPIC loss is small for some z^*, and $\pi(a|s, z^*)$ is near optimal for $R^{int}(\tau, z^*)$, then $\pi(a|s, z^*)$ approaches the optimal policy for the task as specified by R^{ext}. Notably, we *require access to the task reward function R_{ext} to compute the EPIC loss.* We provide some results relaxing this assumption by learning the reward function from data in Table 3. We provide details regarding approximating the EPIC loss in Appendix 5.2.

Computing R^{int} during reward matching During pretraining, for some methods such as [18, 27], we require negative samples in order to compute the variational objective in Equation 2 and avoid a degenerate optimization where all embedded trajectories have high similarity with all skills. However, during selection when skills are fixed, negative sampling amounts to a reward offset which does not impact the task semantics. Furthermore, since we may not in general have access to a large amount of negative samples on a given downstream task, we choose to simplify the objective to the following:

$$\mathcal{R}^{int}(\tau, z) := \text{VLB}(\tau, z) \equiv q_\phi(\tau, z) \tag{4}$$

where q_ϕ is the skill discriminator. This parameterization of the intrinsic reward preserves the alignment semantics of VLB without the normalization by negative samples.

4 Experiments and Analysis

In this section we aim to experimentally evaluate whether IRM improves the adaptation sample-efficiency of skill finetuning on a downstream reinforcement learning task as compared to baselines. For pretraining skills, we experiment with the Contrastive Intrinsic Control (CIC) [18] algorithm. We consider *IRM Random* a version of IRM that randomly samples skills and picks the one with the lowest EPIC loss, *IRM CEM* which selects CEM elites as those skills with the lowest EPIC loss, and *IRM Gradient Descent* which minimizes the EPIC loss using the Adam optimizer and uses backpropagation through the discriminator to regress the optimal skill.

Task	IRM Rand	IRM CEM	IRM GD	Env Roll.	HRL
Fetch Reach Seq Easy	88.1 ± 1.5	**89.5** ± **0.34**	86.7 ± 0.64	80.7 ± 4.7	28.4 ± 31.0
Fetch Reach Seq Hard	**73.4** ± **2.8**	66.4 ± 1.1	**72.4** ± 3.3	62.7 ± 6.1	28.0 ± 30.6
Fetch Barrier Easy	29.3 ± 16.4	49.0 ± 22.1	**57.5** ± **9.5**	65.5 ± 9.0	30.1 ± 20.0
Fetch Barrier Hard	43.3 ± 16.3	48.1 ± 17.8	**88.2** ± **10.4**	65.4 ± 5.6	42.8 ± 18.2
Fetch Tunnel Easy	20.9 ± 14.3	40.8 ± 13.0	49.1 ± 8.4	**52.2** ± **9.1**	32.2 ± 13.9
Fetch Tunnel Hard	25.6 ± 9.4	**58.4** ± **1.8**	35.7 ± 1.3	14.5 ± 4.8	33.7 ± 13.9

Table 2: Rewards on long-horizon manipulation tasks

Environments We design both a goal reaching and a tabletop pushing environment in the OpenAI Gym Fetch environment [5]. For the reaching tasks, the robot arm is tasked with reaching towards one (Reach) or many successive (Reach Seq) specified target locations. For the block pushing (Push)

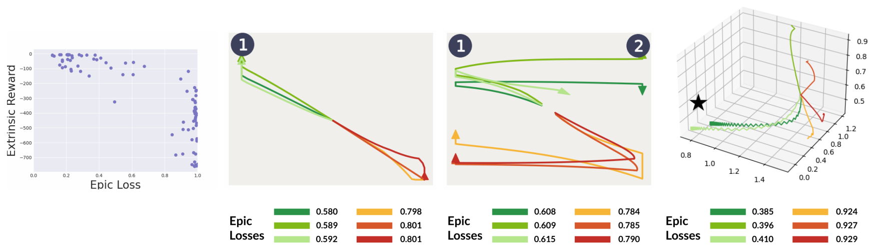


Figure 3: (a) Scatter plot of extrinsic reward vs. EPIC loss. (b-c) Skill trajectories with low and high EPIC losses for planar goal-reaching. (d) Skill trajectories for Fetch Reach.

environments, we introduce obstacles (Barrier, Tunnel) around which the policy must manipulate the object as pictured in Appendix 7. Further task design details are provided in Appendix 5.11.

Baselines We benchmark many conventional finetuning approaches after a single skill pretraining phase of CIC [18]. The *Grid Search (GS)* baseline coarsely sweeps each of 10 skills evenly from the all 0's skill vector to the all 1's skill vector and finetunes the skill which achieves the best evaluation reward over an episode. *Env Rollout* randomly samples 10 skills to evaluate with a rollout and *Env Rollout Cross-Entropy Method (CEM)* uses the episode reward as the metric by which to select elites. *HRL* is a hierarchical reinforcement learning algorithm (Appendix 5.17) that learns a high-level policy to compose the frozen low-level skills for the long-horizon tasks. *Random Skill* selects a skill at random. All baselines use the TD3 [11] RL algorithm.

Evaluation First, we pretrain each RL agent with the intrinsic rewards for 2M steps. We then evaluate the rewards achieved by the selected skill policy but without any RL updates on the task reward. We report results averaged over 3 seeds with standard error bars.

Fetch Tabletop Manipulation On the Fetch Reach task, IRM outperforms or performs similarly to environment-rollout methods while requiring no environment samples to perform skill selection. As shown in Table 1, the random skill policy performs particularly poorly and with very high variance relative to the IRM and environment-rollout based methods. Moreover, appropriate skill selection is required for strong zero-shot performance, as certain skills obtain much higher rewards than others. Next, we evaluate IRM on more complex manipulation tasks involving pushing a block to a goal position. This more complex task similarly benefits from bootstrapping the appropriate pretrained skill policy, evidenced by the performance gap of selection based methods over random skill selection. Although Env Rollout CEM can be a strong interaction baseline for of zero-shot reward, far exceeding a reasonable budget of interactions entirely on skill selection.

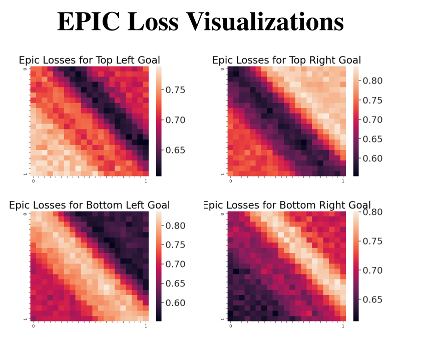


Figure 2: EPIC losses between extrinsic rewards and intrinsic rewards conditioned on the skill vector. We sweep over the 2D skill vector for a pretrained planar agent.

How Can We Understand IRM Optimization? In Figure 3, we verify that EPIC loss is negatively correlated with extrinsic reward on a Planar Goal Reaching task detailed in Appendix 5.15. This provides some evidence that, optimizing for a low EPIC loss can lead to higher environment reward. In Figure 2, we plot EPIC losses between intrinsic rewards and goal-reaching rewards across the 2D continuous skill space. Not only is the loss landscape smooth, which motivates optimization methods like gradient descent, but there is also a banded partitioning of the manifold. Furthermore, the latent skill space is well-structured as different darker-colored partitions of the skill space correspond to the group of skills with low EPIC loss from each task reward. In Figure 3, skills with the lowest EPIC loss receive high extrinsic reward, reaching the goal with high spatial precision. Skills with the highest losses produce the opposite behavior, moving in the direct opposite direction of the goal. In the sequential case, low-EPIC loss skills attempt to reach the 1st goal then the 2nd goal, while high-EPIC loss skills perform the behavior in the inverse order. The intrinsic reward module provides deeper insight into the semantics of skills than the extrinsic rewards obtained by skill policy rollouts.

References

[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *CoRR*, abs/1807.10299, 2018.

[2] David Barber and Felix V. Agakov. The im algorithm: A variational approach to information maximization. In *NIPS*, pages 201–208, 2003.

[3] André Barreto, Rémi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. *CoRR*, abs/1606.05312, 2016.

[4] Kate Baumli, David Warde-Farley, Steven Hansen, and Volodymyr Mnih. Relative variational intrinsic control. *CoRR*, abs/2012.07827, 2020.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function, 2018.

[7] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[8] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *CoRR*, abs/1704.03012, 2017.

[9] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *CoRR*, abs/1710.09767, 2017.

[10] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[11] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[12] Adam Gleave, Michael Dennis, Shane Legg, Stuart Russell, and Jan Leike. Quantifying differences in reward functions. *CoRR*, abs/2006.13900, 2020.

[13] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control, 2016.

[14] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *CoRR*, abs/1611.07507, 2016.

[15] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *CoRR*, abs/1912.01603, 2019.

[16] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[17] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021.

[18] Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. Cic: Contrastive intrinsic control for unsupervised skill discovery, 2022.

[19] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. URLB: unsupervised reinforcement learning benchmark. *CoRR*, abs/2110.15191, 2021.

[20] Xinran Liang, Katherine Shu, Kimin Lee, and Pieter Abbeel. Reward uncertainty for exploration in preference-based reinforcement learning, 2022.

[21] Hao Liu and Pieter Abbeel. Aps: Active pretraining with successor features, 2021.

[22] Eric J. Michaud, Adam Gleave, and Stuart Russell. Understanding learned reward functions. *CoRR*, abs/2012.05862, 2020.

[23] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *CoRR*, abs/1805.08296, 2018.

[24] Marc Pickett and Andrew G Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *ICML*, volume 19, pages 506–513, 2002.

[25] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent, 2022.

[26] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

[27] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020.

[28] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In Sven Koenig and Robert C. Holte, editors, *Abstraction, Reformulation, and Approximation*, pages 212–223, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[29] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[30] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994.

[31] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[32] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *CoRR*, abs/1703.01161, 2017.

5 Appendix

5.1 Equivalent-Policy Invariant Comparison

We can formalize a general notion of reward function similarity by equivalent-policy invariant comparison (EPIC) as established in [12]. EPIC defines a distance metric between two reward functions such that similar reward functions induce similar optimal policies. We consider the case of action-independent reward:

$$D_{\text{EPIC}}(R_A, R_B) = \mathbb{E}_{s_P, s'_P \sim D_P, S_C, S'_C \sim D_C}[D_\rho(C(R_A)(s_P, s'_P, S_C, S'_C), C(R_B)(s_P, s'_P, S_C, S'_C))]. \quad (5)$$

where $D_\rho(X,Y) = \sqrt{\frac{1-\rho(X,Y)}{2}}$ is the Pearson distance between two random variables X and Y, s_P, s'_P are samples from the Pearson distribution D_P, and S_C, S'_C are batches sampled from the Canonical distribution D_C. We compute the Pearson distance over Pearson samples s_P, s'_P, with additional canonicalization with batches S_c, S'_c to ensure invariance over constant shifts and scaling. The canonicalized reward function is defined as:

$$C(R)(s_P, s'_P, S_C, S'_C) = R(s_P, s'_P) + \mathbb{E}[\gamma R(s'_P, S'_C) - R(s_P, S'_C) - \gamma R(S_C, S'_C)] \quad (6)$$

where $R : S \times S \rightarrow \mathbb{R}$ is a reward function and γ is the discount factor [12]. The expectation is taken over the Canonical distribution D_C; for simplicity, we sample these batches S_C, $S'_C \sim D_C$ ahead of time. The canonicalization ensures *invariance to reward shaping* such that rewards that have different shaping but induce similar optimal policies are close in distance. In practice, the final term can be omitted as the Pearson correlation is *invariant to constant shifts and scaling*.

The EPIC psuedometric has typically been used for benchmarking algorithms and evaluating learned reward functions [22, 20]. To the best of our knowledge, IRM is the first to use EPIC in an optimization objective, particularly for complex tasks and larger state dimensions.

5.2 EPIC Sample-Based Approximation

We make a number of sample-based approximations of various unknown quantities in order to concretize the continuous optimization Equation 3 as a tractable loss minimization problem.

Canonical State Distribution Approximation: In order to canonicalize our reward functions, we estimate the expectation over the state and next state distributions with a sample-based average over 1024 samples. These distributions can be entirely arbitrary, though using heavily out-of-distribution samples with respect to pretraining can weaken the accuracy of the approximation. We choose to instantiate a uniform distribution bounded by known workspace constraints for both of these distributions.

Sampling Distribution for Pearson Correlation: We find that generating samples uniformly roughly within the environment workspace bounds, just as with the reward canonicalization, often leads to strong approximations. Furthermore, as both sample generation and relatively inexpensive function evaluation are independent of the online-finetuning phase, we can perform the full skill optimization as a self-contained preprocess to downstream policy adaptation without any environment samples. Rough knowledge of workspace bounds represents some amount of prior environment knowledge. We leave more general options such as sampling from a learned generative model over trajectories encountered during pretraining or sampling from saved pretraining data to future work. We ablate various sampling distribution choices in Table 8 and present the full algorithm in detail in Algorithm 1.

5.3 Reward Matching Ablations

In addition to the EPIC matching metric, we also include additional ablations using Dynamics-Aware Comparison of Learned Reward Functions (DARD) matching. DARD extends EPIC by sampling the states used to compute the EPIC distance from a learned dynamics model to generate more realistic transitions.

Algorithm 1: Intrinsic Reward Matching (IRM)

```
Require: Downstream task $\mathcal{T}$, $D_P$, $D_C$
Require: Pretrained policy $\pi_\theta(a|s, z)$, intrinsic reward $r_{\text{int}}(s, s', z)$, and extrinsic reward
         $r_{\text{ext}}(s, s')$ for $\mathcal{T}$.
Require: Optimization $N_{\text{OP}} = 5000$ steps and finetune for $N_{\text{FT}} = 100K$ steps.
/* Skill Selection of $z^*$ via EPIC Loss                                    */
1 for $N_{OP}$ steps do
2     Sample a batch of Pearson samples $S_P, S'_P \sim D_P, D_P$.
3     Sample Canonical samples $S_C, S'_C \sim D_C, D_C$.
4     for $s_i, s'_i$ in $S_P, S'_P$ do
5         Calculate EPIC Loss as $D_{\text{EPIC}}(r_{\text{int}}(s_i, s'_i, z), r_{\text{ext}}(s_i, s'_i)) =$
            $D_\rho(C_{D_C}(r_{\text{int}})(s_i, s'_i, S_C, S'_C), C_{D_C}(r_{\text{ext}})(s_i, s'_i, S_C, S'_C))$, as in Equation 5.
6     end for
7     Take optimization step on batch with respect to $z$ (gradient descent, CEM step, etc.), as in
        Equation 3.
8 end for
9 Evaluate zero-shot performance and finetune RL agent for $N_{\text{FT}}$ steps with $z^*$ on downstream task
  $\mathcal{T}$.
```

Task	**Fetch Reach**	**Fetch Push**
IRM CEM	95.9 ± 1.0	80.2 ± 2.5
IRM GD	87.5 ± 0.20	73.1 ± 0.48
IRM Rand	92.5 ± 1.1	77.6 ± 2.7
IRM CEM + DARD	91.1 ± 3.0	76.3 ± 2.0
IRM GD + DARD	53.6 ± 23.5	61.6 ± 20.9
IRM Rand + DARD	88.9 ± 4.0	74.5 ± 2.3
IRM CEM + LRF DARD	88.3 ± 1.8	67.2 ± 3.9

Table 3: **Zero-Shot Rewards**. We replace EPIC with DARD. We also compare with learned reward functions (LRFs) to show potential for future work in exploring IRM without known or estimated reward functions.

We also include ablations of DARD using a *learned reward function*, addressing issues where a reward function may not be known beforehand. We include results in Table 3

DARD DARD's dynamics model is trained on 4k environment samples, batch size 1024, two-layer MLP, hidden dimension 256. For Fetch Reach, we train for 2500 grad steps. For Fetch Push, we train for 800 gradient steps.

LRF DARD LRF DARD's dynamics model is also trained on 4k environment samples, batch size 1024, two-layer MLP, hidden dimension 256. We parameterize the task reward function as a 3-layer MLP with a hidden dimension of 32 that computes the scalar-valued reward from inputted state and next-states on the same 4k environment samples. For Fetch Reach, we train for 100 gradient steps. For Fetch Push, we train for 500 gradient steps.

5.4 Generalization to Skill Sequencing

Many realistic downstream tasks derive additional complexity from temporally extended planning horizons. In contrast to hierarchical reinforcement learning (HRL) approaches, which aim to stitch together pretrained skills at the policy level with a higher-level manager policy, we can extend the task matching framework of IRM to efficiently solve the problem of skill sequencing, entirely doing away with the manager policy. Consider the long-horizon setting where we have a sequence of reward functions over task horizon H. Central to the finetuning problem is determining over what time intervals should potentially different pretrained skills be selected. In this work, we predetermine a fixed skill horizon $\lfloor H/N \rfloor$, where N is the number of rewards. This skill horizon could also be specified as a parameter and learned from the task reward signal.

Next, to perform skill selection over each time interval, we perform the IRM algorithm in parallel or sequentially for each reward. We note the key assumption that IRM requires access to the reward functions for each of the subtasks. For example, for a sequential goal reaching task, we divide the

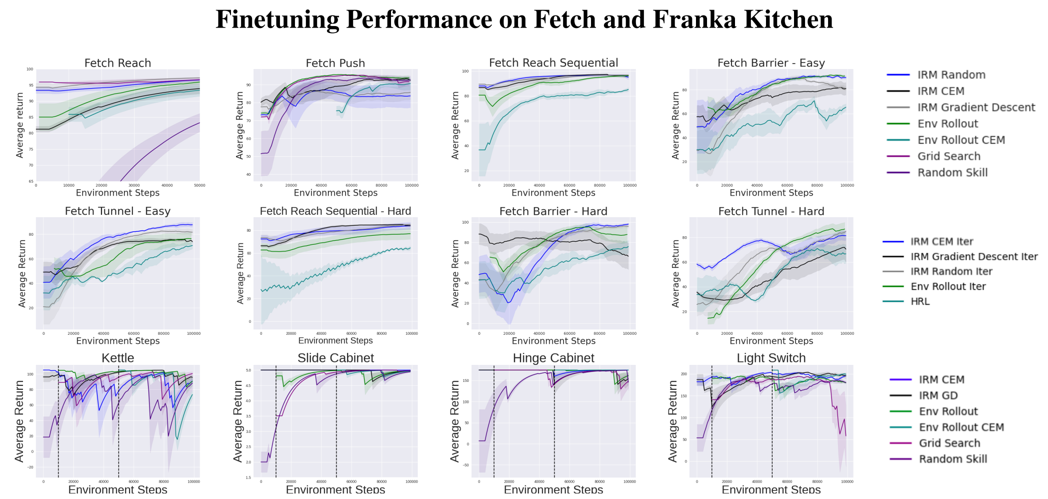


Figure 4: The performance gap between the IRM skill selection methods and random skill selection evidences the sample efficiency gains to be had from bootstrapping a pretrained policy with task-level semantics similar to the task reward. IRM-based methods select optimal skills with no environment interaction and consequently finetune efficiently. **Top:** Fetch Reach and Block Push tasks. **Middle:** Long-horizon Fetch Reach, Fetch Barrier, and Fetch Tunnel tasks across difficulties. **Bottom:** Fetch Kitchen Manipulation Tasks

episode into N segments for each of the N goals and corresponding rewards. We perform the IRM skill selection algorithm for each reward to select the optimal skill over each interval. After selecting the skills, we freeze our selections and finetune the skill policies jointly.

5.5 Extensions and Ablations

Long-Horizon Manipulation Building on the results in Section 4, we demonstrate that IRM fully generalizes to solving long-horizon tasks in the setting of tabletop manipulation. During the unsupervised pretraining phase, skill discovery methods can acquire useful skills such as directional block pushing or pushing the block to certain spatial locations. We show that IRM can intelligently select a sequence of such skills to finetune via reward matching, avoiding learning a hierarchical manager policy that finetunes at the policy level.

For the Fetch Reach Sequential tasks, we consider an extended horizon where the agent reaches a sequence of goals in a particular order. For the Fetch barrier tasks, we consider the environments such as depicted in Figure 7, where the agent navigates around a barrier introduced during the finetuning phase. The tunnel environments consists of a goal enclosed in a tunnel; the agent navigates around and into the tunnel to complete the task. We compare IRM methods to an environment rollout baseline (Env Seq) and a hierarchical RL baseline (HRL). The 'IRM Seq' methods select skills based on each defined sub-task's reward function according to the IRM optimization scheme. 'Env Seq' chooses the best combination of skills based on extrinsic reward from rollouts. 'HRL' is initialized with random skills and simultaneously optimizes a manager policy over skills and the skill policies themselves. In both settings, IRM methods outperform the environment rollout method and in some cases the HRL method in identifying (Table 2) and finetuning skills (Figure 4). Implementation details are provided in Appendix 5.16.

Matching Metric Ablations We validate the importance of employing the EPIC pseudometric for formulating the matching loss by ablating its contribution against more naive selections in Table 4. L1 and L2 losses are common metrics in supervised regression problems but are poor choices for comparing task similarity with rewards. Moreover, rewards can have arbitrary differences in scaling and shaping that L1 and L2 are not invariant to. To strengthen these comparisons, we include a learned reward scaling parameter for L1 and L2 and similarly observe that EPIC is a superior matching metric.

Skill Discovery Algorithm Ablations IRM is fully general to any mutual information maximization based, RL pretraining algorithm as shown in Table 5. We validate on the Fetch Reach task that IRM CEM and IRM Rand convincingly outperform all episode rollout baselines in zero-shot episode reward.

5.6 Related Work

Several works including [27, 7, 1, 14, 4, 8, 18] employ mutual information maximization for skill pretraining. While [18] leverages coarse grid search to select skills for downstream RL, methods such as [27] instead plan through a learned skill dynamics model at finetuning time. Our approach is similar in that it leverages pretraining model components other than the policy to guide skill selection. However, rather than generating a reward maximizing plan through possibly complex, learned environment dynamics, we instead look to match a policy to the task reward directly through a pretrained discriminator.

Reward Matching	IRM CEM
IRM	**21.0** $\pm$ 0.75
L1	8.71 $\pm$ 0.70
L2	7.87 $\pm$ 0.86
L1 + Learn Scale	5.51 $\pm$ 1.9
L2 + Learn Scale	3.95 $\pm$ 2.2

Table 4: Reward matching metric ablation

In the context of sequential finetuning, [4, 7] employ hierarchical RL to chain pretrained skills with a manager policy requiring additional environment interactions. Works on such HRL methods include [23, 9, 32, 16] and more classic approaches like [29, 28]. By contrast, we demonstrate that the intrinsic reward matching framework can be extended to choose skill sequences without reliance on environment samples. The successor features line of work also adopts a unified view of skill-based RL. Such work relies on the assumption that arbitrary rewards can be parameterized linearly in some learned features and some task vector as in [21, 3]. Our approach relaxes this assumption to the fully general setting by instead searching for a pretrained task with minimal proximity to an arbitrarily parameterized task reward.

5.7 Background and Notation

Markov Decision Process: The goal of reinforcement learning is to maximize cumulative reward in an uncertain environment it interacts with. The problem can be modelled as a Markov Decision Process (MDP) defined by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition probability distribution, r is the reward function and γ is the discount factor.

Unsupervised Skill Discovery: In competence-based unsupervised RL the aim is to learn skills that generate diverse and useful behaviors [7]. The broad aim is to learn policies that are skill-conditioned and generalizable. Formally, we also learn skills $z \in \mathcal{Z}$ and take actions according to $a \sim \pi(\cdot|s, z)$. As an illustrative example, applying this formalism to the Mujoco Walker domain, we might hope to find a skill-conditioned policy and skills $z_{\text{walk}}, z_{\text{run}}$ such that $\pi(\cdot|s, z_{\text{walk}})$ makes the agent walk, while $\pi(\cdot|s, z_{\text{run}})$ makes it run. Further, if we allow for continuous skills, we can also imagine being able to use the policy to "jog" at different speeds by interpolation the z_{walk} and z_{run} skills. That is, taking $z_{\text{jog}}^{\alpha} = \alpha \cdot z_{\text{walk}} + (1-\alpha) \cdot z_{\text{run}}$ should, intuitively, yield a policy $\pi(\cdot|s, z_{\text{jog}}^{\alpha})$ that makes the agent jog at speed dictated by the parameter α.

Finetuning Pretrained Skills: With a skill-conditioned policy $\pi(\cdot|s, z)$, an agent needs to infer which skill to index for a downstream task (e.g. identifying if it needs to use z_{walk} or z_{run}) during finetuning. This is a relatively under-explored area, with the most universal approach being a coarse, discretized grid search. Least squares regression has also been investigated in the context of successor features [21].

5.8 Competence-Based Skill Discovery

Competence-based skill discovery algorithms aim to maximize the mutual information between trajectories and skills:

$$I(\tau; z) = \mathcal{H}(z) - \mathcal{H}(z|\tau) = \mathcal{H}(\tau) - \mathcal{H}(\tau|z) \tag{7}$$

Since the mutual information $I(s; z)$ is intractable to calculate, in practice, competence-based methods maximize a variational lower bound. Many mutual information maximization algorithms, such as Variational Intrinsic Control [13] and Diversity is All You Need [6], use the estimate $I(\tau; z) = \mathcal{H}(z) - \mathcal{H}(z|\tau)$. Other competence-based methods, such as Dynamics-Aware Unsupervised

Discovery of Skills [26], Active Pretraining with Successor Features [21], and Contrastive Intrinsic Control (CIC) [18], maximize a lower bound for $\mathcal{H}(\tau) - \mathcal{H}(\tau|z)$.

While the decompositions of the mutual information objective are equivalent, algorithms make different design choices regarding how to approximate entropy, represent trajectories, and embed skills. These choices affect the distillation of skills: for instance, without explicit maximization of $\mathcal{H}(\tau)$ in the decomposition of mutual information, behavioral diversity may not be guaranteed when the state space is much larger than the skill space [18].

5.9 CIC

Contrastive Intrinsic Control (CIC) [18] is a state of the art algorithm for competence-based skill discovery. CIC maximizes a lower bound for $I(\tau; z) = \mathcal{H}(\tau) - \mathcal{H}(\tau|z)$ through a particle estimator for $\mathcal{H}(\tau)$ and a contrastive loss from Contrastive Predictive Coding (CPC) [31] for $\mathcal{H}(\tau|z)$. The lower bound for $I(\tau; z)$ is:

$$I(\tau; z) \geq F_{\text{CIC}}(\tau; z) := \mathcal{H}_{\text{particle}}(\tau_i) + \mathbb{E}\left[q_\phi(\tau_i, z_i) - \log \frac{1}{N}\sum_{j=1}^{N} \exp(q_\phi(\tau_j, z_i))\right] \tag{8}$$

where $\mathcal{H}_{\text{particle}}(\tau) \propto \sum_{i=1}^{n} \log ||h_i - h_i^*||$, h_i^* is the k-Nearest Neighbors embedding, N_k is the number of k-NNs used to approximate entropy, and $N - 1$ is the number of negative samples.

5.10 DADS

We additionally use Dynamics-Aware Unsupervised Discovery of Skills (DADS) [27] for skill discovery, as it is one of the few skill discovery algorithms to successfully scale up to continuous skills. DADS maximizes a lower bound for $I(\tau; z) = \mathcal{H}(\tau) - \mathcal{H}(\tau|z)$ through learning skill-conditioned transition distributions. The lower bound for $I(\tau; z)$ is:

$$I(\tau; z) \geq F_{\text{DADS}}(\tau; z) := \log \frac{q_\phi(s'|s, z)}{\sum_{i=1}^{L} q_\phi(s'|s, z_i)} + \log L \tag{9}$$

Intr. Rew.	**IRM CEM**	**IRM GD**	**IRM Rand**	**Env Roll.**	**Env CEM**	**GS**	**Rand**
DADS	**83.4** ± **2.19**	69.9 ± 2.22	77.2 ± 3.83	74.6 ± 5.15	70.3 ± 5.38	68.9 ± 2.81	28.3 ± 13.5

Table 5: Zero-shot rewards for DADS skill discovery algorithm.

For our experiments, we reimplement the on-policy DADS algorithm in PyTorch. We follow the default hyperparameters and train for 20 million environment steps, per [27].

5.11 Environment Details

For our Fetch Reaching environment, we use the Gym Robotics Fetch environment [5]. We set the time limit to 200. For the fetch push environment, we partition the continuous action space into 4 actions, which involve pushing the block forward, backward, left, and right. We set the time limit to 10 for skill learning.

We evaluate sequential skill selection on Fetch Reach and Fetch Push. For the Fetch Push agent, we have a Barrier and Tunnel environment, each with 3 waypoints, depicted in Figure 7. We fix a time horizon of 30 pushes per waypoint. For Fetch Barrier, the easy task consists of the first two waypoints (L shape), and the hard task consists of all three (U shape). For Fetch Tunnel, the easy task consists of the last two (into the tunnel), and the hard task consists of all three waypoints (side of the tunnel, and then into the tunnel). For Fetch Reach, we consider 2 waypoints and a time horizon of 25 for each waypoint.

Our plane environment is a 2D world with observations in [-128, 128] x [-128, 128] and continuous actions in [-10, 10] x [-10, 10].

The Franka Kitchen Environment consists of a 9 dof Franka Arm in a Kitchen Environment with a variety of rigid and articulated objects. The benchmark specifies several goals with rewards for

Task	IRM CEM	IRM GD	Env Roll.	Env CEM	GS	Rand
Kettle	**105** $\pm$ 0.00	96.2 $\pm$ 6.3	105 $\pm$ 0.27	105 $\pm$ 0.00	89.1 $\pm$ 0.00	18.7 $\pm$ 26.4
Slide Cabinet	**5.00** $\pm$ 0.00	5.00 $\pm$ 0.00	4.81 $\pm$ 0.13	5.00 $\pm$ 0.00	3.51 $\pm$ 0.00	2.00 $\pm$ 0.34
Hinge Cabinet	**175** $\pm$ 0.00	175 $\pm$ 0.00	175 $\pm$ 0.00	175 $\pm$ 0.00	175 $\pm$ 0.00	7.02 $\pm$ 75
Light Switch	182 $\pm$ 9.3	187 $\pm$ 12.9	190 $\pm$ 13.4	**208** $\pm$ 0.32	141 $\pm$ 0.00	53.7 $\pm$ 31

Table 6: Zero-shot rewards for Franka Kitchen tasks

reaching those goals. The 7d action space includes the end-effector 3d translation, 3d rotation, and a gripper open/close dimension. The state includes the joint positions corresponding to each dof of the arm in addition to the joint positions of articulated bodies and 7-dimensional position and orientation information for rigid objects in the scene. The Kettle task involves moving the kettle to a target burner location on the stove. The Hinge and Slide Cabinet tasks involve opening cabinet drawers with revolute and prismatic joints respectively. The Light Switch task involves flipping the switch for the stove light. We use an episode length of 1000 for solving the tasks.

5.12 Pretraining Hyperparameters

For the Fetch Reach environment, we use a skill dimension of 8 and a discriminator MLP hidden dimension of 64. We use an alpha value of 0 for entropy weighting. For the Fetch Push environment, we use a skill dimension of 16 and a discriminator MLP hidden dimension of 16. We use an alpha value of 0 for entropy weighting. For the Franka Kitchen environment we use an alpha value of 0.1 for entropy weighting along with a skill dimension of 2 and an MLP hidden dimension of 64. For all environments, we use a replay buffer size of 100k.

5.13 Intrinsic Reward Matching and Environment Rollout Baseline hyperparameters

IRM CEM and Env Rollout CEM are trained for 5 iterations with 1000 samples at each iteration and 100 elites selected each iteration. Env Rollout CEM consumes the entire downstream finetuning budget on just skill selection. For illustrative purposes, we start its plot at 50k steps to show that finetuning still occurs, however, sample-inefficiency suffers due to excessive rollouts for skill selection. This problem only worsens for long time horizons. IRM Gradient Descent is trained for 5000 steps with a learning rate of 5e-3 and initialized at the skill vector of all 0.5s. IRM Random selects 100 random skills. Env Rollout trials 10 random skills for a fully episode. Grid Search coarsely trials 10 skills from the skill of all 0s to the skill of all 1s as in [19].

5.14 Franka Kitchen Manipulation

We evaluate IRM on 4 manipulation tasks in the Franka Kitchen benchmark [10]. We make no changes to the default environment rewards for each task, which do not include any shaping for reaching to the correct object, and thus present a harder exploration problem. We similarly observe favorable zero-shot performance of the IRM method, in the case of hinge cabinet and slide cabinet, *solving the task zero-shot without any finetuning (Table 6, Figure 4). This zero-shot adaptation behavior presents a compelling means for agents to immediately deploy optimal policies in highly sample-sensitive applications.*

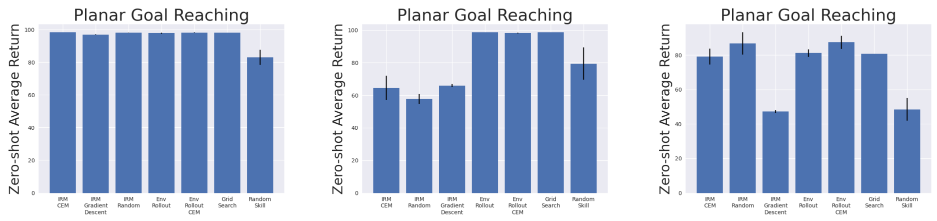
5.15 Planar Goal Reaching

The planar goal reaching task consists of a simple 2D plane with a point with a 2D Cartesian state space that can displace in the x and y coordinates with a 2D action space. Skills learned tend to span the 2D space reaching to diverse locations distributed broadly across the environment. We show some sample zero-shot skill selection results over three different skill dimensions in Figure 5.

5.16 Sequential Skill Selection

For sequential skill selection, we compare IRM Sequential and Environment Sequential skill selection. IRM Sequential consists of an iterative process. The first skill is chosen entirely free of environment samples, exactly identical to the single-skill tasks. Once the first skill is chosen, we roll out a

Zero-Shot Performance for Planar Goal Reaching



(a) Skill dimension 2 (b) Skill dimension 8 (c) Skill dimension 16

Figure 5: Zero-Shot Returns for Planar Goal Reaching averaged over 5 seeds

Skill Dim	**IRM CEM**	**IRM GD**	**IRM Rand**	**Env Roll.**	**Env CEM**	**GS**	**Rand**
8	21.1 $\pm$ 0.51	15.7 $\pm$ 1.61	18.9 $\pm$ 0.18	18.4 $\pm$ 0.18	18.8 $\pm$ 0.48	17.9 $\pm$ 0.101	13.5 $\pm$ 1.85
16	17.4 $\pm$ 1.30	14.6 $\pm$ 0.63	18.8 $\pm$ 0.26	22.7 $\pm$ 0.83	23.1 $\pm$ 0.36	14.0 $\pm$ 0.19	11.2 $\pm$ 2.32
32	20.1 $\pm$ 0.54	22.537 $\pm$ 0.25	19.8 $\pm$ 0.14	22.2 $\pm$ 0.58	21.5 $\pm$ 0.67	24.0 $\pm$ 0.12	19.9 $\pm$ 0.67
64	21.9 $\pm$ 0.48	1.68 $\pm$ 0.069	20.9 $\pm$ 0.74	22.5 $\pm$ 0.70	21.6 $\pm$ 0.89	18.2 $\pm$ 0.059	13.3 $\pm$ 2.15

Table 7: IRM methods and environment rollout methods ablated over multiple skill dimensions on Fetch Push

trajectory with the skills we have chosen so far and use the latter half of the trajectory as the Pearson samples for our EPIC loss. We use Gaussian noise with variance 1 for our Canonical samples as described in Appendix 5.19.2. At each step of the skill selection process, we use the corresponding IRM optimization methods.

For our Environment Sequential skill selection method, we select skills iteratively as well. For each waypoint or subtask, we randomly sample N skills and commit to the best, where $N = 10/\text{n\_subtasks}$.

5.17 Hierarchical Reinforcement Learning Baseline

In order to validate the benefits of IRM's offline skill selection, we compare against a baseline that leverages a conventional hierarchical RL algorithm to solve long-horizon, sequential tasks. We instantiate a TD3 manager agent that outputs into a skill action space from state input at a temporally abstract timescale. As in the IRM setup, this timescale is fixed to align with the changes in reward to encourage the manager to change its skill prediction according to the change in the reward semantics. The manager's is then inputted to the low-level pretrained skill policy which is rolled out over many steps with the skill fixed. Both the manager policy and the low-level policy weights are updated during finetuning. The manager agent is randomly initialized such that its initial skill prediction is random.

5.18 Compute

We thank NVIDIA for providing compute resources through the NVIDIA Academic DGX Grant. We used a cluster of 8 NVIDIA A100s for experimentation.

5.19 Additional Ablations

5.19.1 Skill Dimension

We ablate skill dimension and evaluate the zero-shot performance of all skill selection methods. IRM's performance generally increases with increased skill dimension despite discriminator overfitting issues associated with larger skill spaces. The IRM GD learning rate is chosen as 5e-3 for all experiments in this work and is not tuned at all. Such likely explains the divergence of the 64 dimensional result.

5.19.2 Pearson & Canonical Distributions

We experiment with many ways to approximate the Pearson and Canonical distributions. We defined Full Random to be our uniform samples from a reasonable estimate of the upper and lower bounds for each dimension of the state. For our planar environment, the bounds are defined explicitly and thus known; for more complex environments, we estimate the bounds. For example, for a tabletop manipulation workspace, we sample 2-dimensional block positions uniformly within the rectangular plane of the table surface. In practice, IRM is fairly robust to the distributions, though there are subtleties that emerge in the various choices for the Pearson and Canonical distributions. For instance, we also ablate a Uniform(0,1) distribution, which generally performs much worse, due to lack of state coverage for most environments. For the Canonical distribution, we also approximate samples by perturbing the Pearson samples by ϵ sampled from a Gaussian distribution. We experiment with hyperparameters of variance, which may be adjusted based on the environment. For our sequential IRM method, we use this Canonical distribution to ablate on-policy samples.

Pearson Distribution	**Canonical Distribution**	**IRM CEM**
Full Random	Full Random	20.341 $\pm$ 0.306
Full Random	Uniform(0,1)	16.343 $\pm$ 0.708
Full Random	$\epsilon \sim \mathcal{N}(0, 1)$	21.191 $\pm$ 0.629
Full Random	$\epsilon \sim \mathcal{N}(0, 0.1)$	21.027 $\pm$ 0.419
Uniform(0,1)	$\epsilon \sim \mathcal{N}(0, 1)$	5.905 $\pm$ 3.157
Uniform(0,1)	$\epsilon \sim \mathcal{N}(0, 0.1)$	2.851 $\pm$ 0.605

Table 8: EPIC Loss Sampling Distribution Ablations.

None of the distributions ablated above require on-policy environment samples. It *is* possible to use on-policy samples for the state distributions, and we choose to do so for our sequential IRM method, as previous skill rollouts may provide useful Pearson samples for the subsequent skill selection. Note that while on-policy Canonical samples are possible, they are incredibly expensive and require access to the environment simulator, so we focus on other choices of distributions.

5.19.3 Sparse Reward Ablation

We ablate our planar EPIC Loss visualizations with sparse rewards. Instead of a well-shaped goal-reaching reward, we use sparse rewards based on the tolerance to the goal. We define the tolerance as the radius the agent must be within if our 2d planar environment is scaled to [0, 1] x [0, 1]. With a very sparse reward, we show that EPIC losses are largely uninformative. However, by slightly relaxing the tolerance, we show a much better shaped EPIC loss landscape that bears similarity to that of Figure 2. Thus, while our method is dependent on access to extrinsic rewards, and ideally, shaped rewards, we show that the EPIC loss landscape over sparse reward landscapes with sufficient tolerance can be optimized.

5.20 Limitations

We acknowledge a number of limitations of our approach. IRM relies on samples of the state, roughly within workspace boundaries as well as access to an external reward function, ideally well-shaped, which trades off with IRM's reduced reliance on environment interactions. In order to obtain realistic image samples to compute the EPIC loss, an agent could learn an expressive generative model over the image states obtained during pretraining and sample from the model to generate diverse and realistic sampled states. For learning unknown state-based rewards, the agent could additionally learn an image-reward model by regressing the rewards encountered during exploration [15]. This represents an exciting direction for future work.

EPIC Loss Visualizations

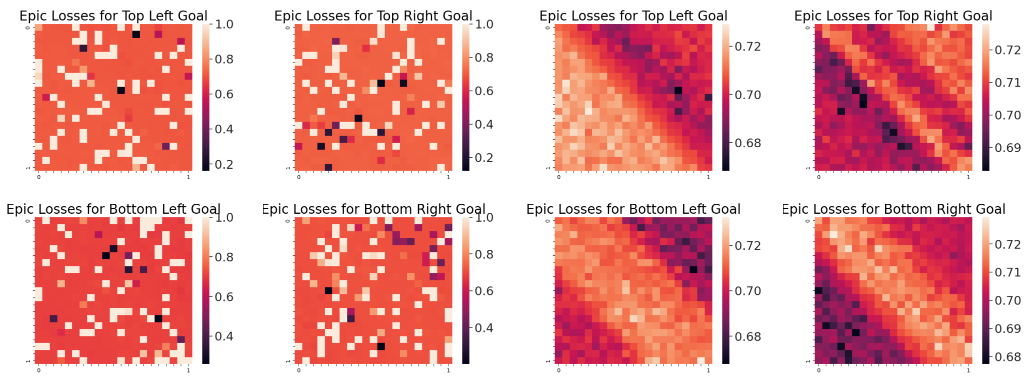


Figure 6: We examine EPIC losses between extrinsic rewards and intrinsic rewards conditioned on the skill vector. We sweep across the 2D skill vector for a pretrained planar agent. Left: Sparse goal-reaching reward with tolerance of 0.03. Right: Sparse goal-reaching reward with tolerance of of 0.07.

Fetch and Franka Kitchen Environments

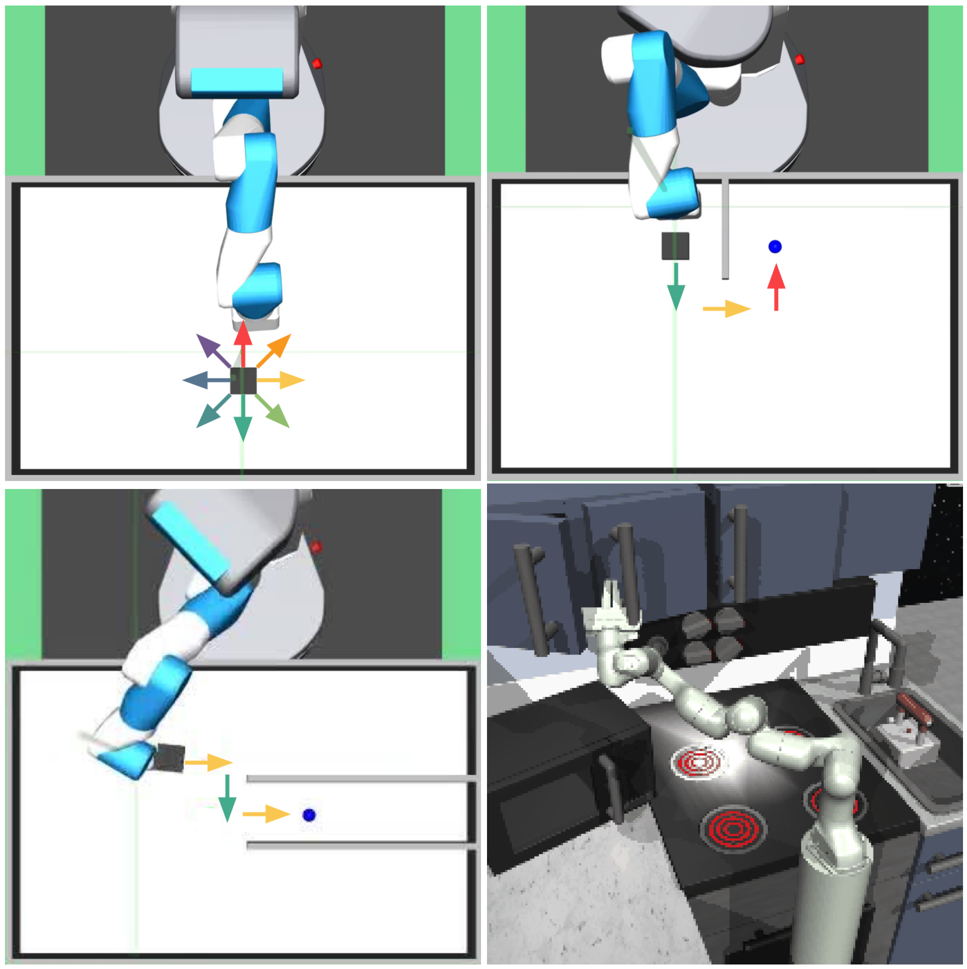


Figure 7: In our Fetch Push environment, we discover skills that move the block in different directions. Downstream tasks may involve simple goals or more distant goals that require composition of multiple skills across an extended time horizon and around obstacles. In Franka Kitchen, the agent discovers skills that interact with various rigid and articulated objects in the scene, which are then leveraged to accomplish diverse tasks.