Appendix A Expanded Related Work

Traditional formulations of imitation learning [8, 1, 9] assume access to a corpus of expert demonstration data which includes both state and action trajectories of the expert policy. In the context of third-person imitation learning, including when learning from expert agents with different embodiments, obtaining access to ground-truth actions is difficult or impossible.

Inferring expert actions. To address this issue, several approaches either try to infer expert actions [10, 11, 12, 32] – for example by training an inverse dynamics model on agent interaction data [10] – or employ forward prediction on the next state to imitate the expert without direct action supervision [13]. In the case of [33], a video-based action classifier trained on a large-scale human activity dataset is leveraged to provide rewards for single-task RL policies, which are then used to provide expert state-action pairs for multi-task behavior cloning. While these methods successfully address learning from observation-only demonstrations, they either do not support skill transfer to different policy embodiments at all, or they cannot take advantage of multiple embodiments in order to improve generalization to unseen policy configurations. We explicitly address these problems in this work.

Imitation via learned reward functions. In contrast to imitation via supervised methods, such as BCO [10], a recent body of work [4, 5, 14, 6, 15, 34] has focused on learning reward functions from expert video data and then training RL policies to maximize this reward. In [4], the authors combine ImageNet pre-trained visual features with an L2-norm distance reward to match policy and expert observations in a latent feature space. In their follow-up work [5], the reward is computed in a viewpoint-invariant representation that is self-supervised on video data. While both these methods are compelling in their use of cheap unlabeled data to learn invariant rewards, the use of a time index as a heuristic for defining weak correspondence is a constraining limitation for tasks that need to be executed at different speeds, or are not strictly monotonic (e.g., have ambiguous sub-task ordering). In [14] a dense reward is learned via unsupervised learning on YouTube data and the authors make no assumption about time alignment. However, in their work, the expert and learned policy are executed in the same domain and embodiment, an assumption we relax in our work. Framed in a multi-task learning setting, [16] propose training policies with morphologically different embodiments first on a similar set of proxy tasks, in order to learn a latent space to map between domains, and then sharing skills on a held-out task from one policy to another. A time-index heuristic is used to define a metric reward when performing RL training of the new task. In our work, the learned embedding finds correspondences in a fully-unsupervised fashion, without the need for such strict time alignment. In [17], a small sub-set of states is human labeled for goal success and a convolutional network is then trained to detect successful task completion from image observations, where on-policy samples are used as negatives for the classifier. By contrast, our learned embedding encodes task progress in its latent representation without the use of expensive human labels.

Imitation via domain adaptation. An additional category of approaches to third-person imitation learning are those that perform domain adaptation of expert observations [18, 19, 20, 21]. For instance, in [18] a CycleGAN [22] architecture is used to perform pixel-level image-to-image translation between policy domains, which is then used to construct a reward function for a model-based RL algorithm. A similar model-free approach is proposed in [19]. In [20], a generative model is used to predict robot high-level sub-goals conditioned on a third-person demonstration video, and a lower-level control policy is trained in a task-agnostic manner. Similarly, [21] uses high level task conditioning from zero-shot transfer of expert demonstrations, but they use KL matching to perform both high and low-level imitation. In contrast to these methods, the unsupervised TCC alignment in this work avoids performing explicit domain adaptation or pixel-level image translation by instead learning a robust and invariant feature space in a fully offline fashion.

Inspired by maximum entropy inverse RL [35, 36] and generative adversarial networks [37], the seminal GAIL algorithm [38] performs distribution matching between the expert and policy's state-action occupancy via an adversarial formulation; a discriminator is trained with on-policy samples, which is then used as a reward in an RL framework. Many recent works [39, 40, 41, 42, 43] build upon GAIL in order to perform observation-only imitation learning using state-only occupancy matching [39], domain adaptation via domain confusion [40, 41], and state-alignment using a variational autoencoder next-state predictor [42]. Likewise, the algorithm proposed in [15] combines a metric learning loss that uses temporal video coherence to learn a robust skill representation with an entropy-regularized adversarial skill transfer loss. Finally, the authors of [44] propose an adversarial formulation for learning across domains with dynamics, embodiment and viewpoint mismatch. In contrast to these methods, our unsupervised reward is robust to domain shift without requiring online fine-tuning or the additional complexity of dynamic reward learning.

Reinforcement learning with demonstrations. Recent work in offline-reinforcement learning [6] explicitly tackles the problem of policy embodiment and domain shift. Their method, Reinforcement Learning from Videos (RLV), uses a labelled collection of expert-policy state pairs in conjunction with

adversarial training to learn an inverse dynamics model jointly optimized with the policy. In contrast, we avoid the limitation of collecting human-labeled dense state correspondences by using a self-supervised algorithm (i.e., TCC [7]) which uses cycle-consistency to automatically learn the correspondence between states of two domains. We also show that this formulation improves generalization to unseen embodiments. Since the problem setup is similar to ours, we also compare to their method as a baseline.

Appendix B X-MAGICAL Benchmark Details

In this section, we provide more information about our X-MAGICAL benchmark, including a description of the task, an overview of the different embodiments and details regarding horizons, the success metric and the environment reward. We encourage the reader to read [24] for an in-depth description of the base MAGICAL benchmark.

B.1 Action and Observation Space

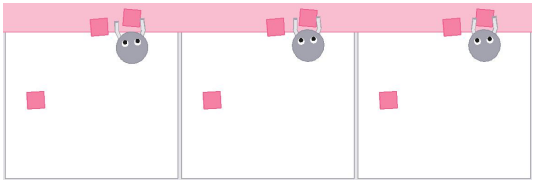

Figure 8. A stack of three image observations for the *gripper* embodiment.

We use a continuous action space for our *Sweeping* task. All embodiments have a 2D action space with the exception of the *gripper* agent which has an additional degree of freedom to open and close its arms. The first degree of freedom is for longitudinal movement (forward/backward), the second degree of freedom is for angular movement (left and right rotation) and the third degree of freedom, if applicable, is a gripping action (push fingers closed/allow fingers to open).

x-MAGICAL provides both state and image-based observations. The state observations are used as input to the RL policies whereas the pixel observations are used by the pretrained encoder to generate rewards. The state vector contains the (x,y) position of the agent, $(\cos\theta,\sin\theta)$ where θ is the agent's 2D orientation, and for each of the three debris: its (x,y) position, its distance to the agent and its distance to the goal zone. For the RGB image, we employ an allocentric, top-down perspective (Figure 8) with full view of the workspace. Similar to MAGICAL, we use an 8Hz control rate, thus with a frame stacking value of 3, this corresponds to roughly 0.3 seconds of interaction.

B.2 Detailed Task and Embodiment Description

In the *Sweeping* task, the robotic agent must push 3 debris blocks to the pink zone at the top of the workspace. The agent's position is constrained to always spawn below the debris. Both the agent's position and the debris positions are randomized at every environment reset. Specifically, we sample the same y-coordinate for all three debris, then randomly space them out from each other (different x-coordinate). The horizon for the *longstick* agent is $H=50$ time steps since it can solve the task much faster than the other embodiments thanks to its morphology. The horizon for all other embodiments is $H=100$. The ground-truth environment reward is defined as $\frac{1}{3}\cdot\sum_{i=1}^{3}\mathbb{1}\{d_i\in G\}$, i.e., the fraction of total debris present inside the goal zone G.

In Figure 9, we provide a film strip demonstration of each embodiment solving the *Sweeping* task with a plot of the environment reward as a function of time. For this visualization specifically, we manually teleoperate each agent and disable the environment horizon limit.

B.3 Demonstration Data

To collect demonstration data for each embodiment, we train an oracle policy with SAC using the ground-truth environment reward. We then rollout the policy in the environment, discarding any potentially unsuccessful demonstrations, until we are left with 1000 demonstrations per embodiment. A comprehensive overview of the hyperparameters used for reinforcement learning are detailed in Table 7 in Appendix G.2.

Appendix C X-REAL: A Real-World Cross-Embodiment Dataset

To test reward learning in the real world on more challenging manipulation tasks, we collect a real-world dataset named X-REAL (Cross-embodiment Real-world demonstrations), which contains 93 demonstration videos of different embodiments (manifested as different manipulator end-effectors) solving the same manipulation task in the real-world: *transferring five pens to two cups consecutively*. This is a multi-step

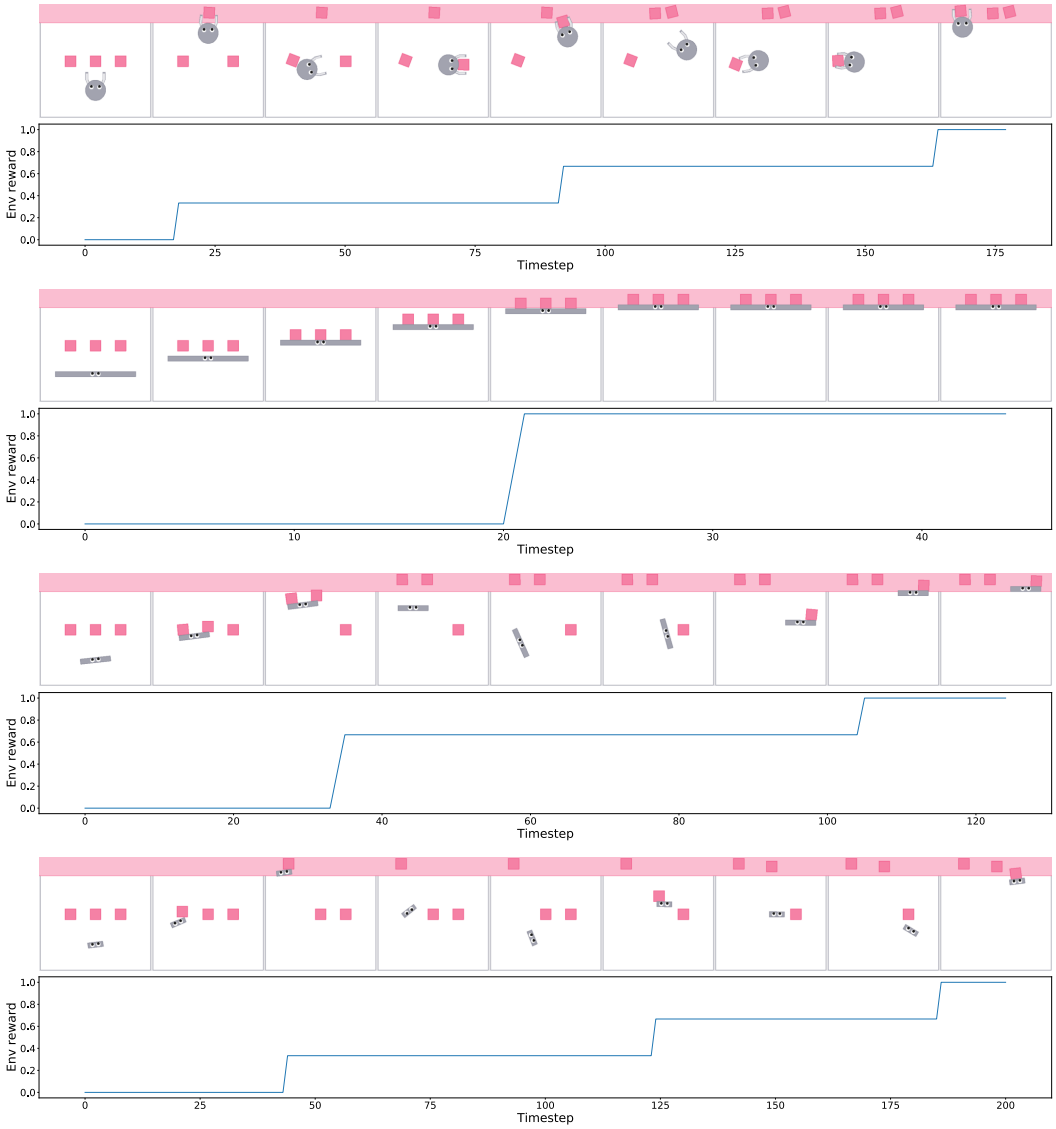

Figure 9. A film strip of the *gripper*, *longstick*, *mediumstick* and *shortstick* embodiments (vertical order) solving the Sweeping task with a corresponding visualization of the ground-truth environment reward.

Embodiment	**Demo Length Stats (seconds)**
1 Hand 5 Fingers	10.8±1.8
2 Hands 2 Fingers	11.8±1.2
1 Hand 2 Fingers	14.9±1.4
Ski Gloves	20.0±3.9
Kitchen Tongs	21.1±3.1
Lobster Hands (costume)	21.3±3.8
Tweezers	27.8±2.3
RMS Grabber Reacher	29.9±3.2
Irwin Quick-Grip Clamps	38.1±9.2

Table 2. Mean ± Std. Dev. of Demonstrations Lengths for Different X-REAL Embodiments

manipulation task where the pens on the table need to be lifted to one cup and then moved again to a separate cup. The different end-effectors consist in a human hand as well as six tools purchased from Amazon and displayed in Figure 10. In contrast to the dataset used in Section 5.3, there is visual diversity among the different end-effectors in X-REAL, and there is also significant variation in how the task is solved and how long it takes to solve the task. Some end-effectors (e.g., tweezers) can only move one pen at a time, while others (e.g., human hand with five fingers) can move all pens at once. Additionally, the demonstrations are not collected in a constrained fashion that tries to mimic the robot. We report the mean and standard deviation of demonstration lengths for each embodiment in Table 2. Note the variation in demo lengths across different embodiments.

Figure 10. Embodiments in the X-REAL dataset, ordered by their appearance in Table 2.

C.1 Data Collection & Hardware Setup

The hardware and data collection setup is shown in Figure 11. We use a GoPro Hero8 mounted on a tripod to record the demonstrations and use voice commands to efficiently start and stop the video recordings. The Hero8 records RGB images with a resolution of 1920×1080 at 30 frames per second.

C.2 Reward Learning from Real-world Multi-Cross-Embodiment Demonstrations

In this section, we learn reward functions from videos in X-REAL, and demonstrate that our method is capable of handling the visual complexities of the real world without requiring annotations of end-effectors, objects, or their states. We train the encoder on all embodiments in the training set and present examples of the learned XIRL rewards on video demonstrations from the validation set in Figure 12. Specifically, we visualize two embodiments: the *RMS Grabber Reacher* (top row) and the human *1 Hand 5 Fingers* (bottom row) and for each embodiment, we show both a successful and unsuccessful trajectory.

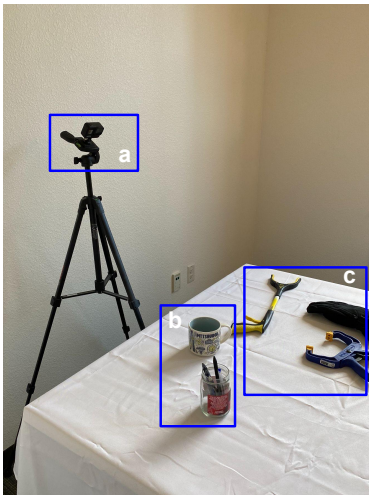

Figure 11. **Hardware setup.** a) Tripod-mounted GoPro Hero8, b) Cup, Mug and Pens from Task c) Example end-effectors used.

In the top row, both the successful and unsuccessful demonstrations follow a similar trajectory at the start of the task execution. The successful one nets a high reward for placing the pens consecutively into the mug then into the glass cup, while the unsuccessful one obtains a low reward because it drops the pens outside the glass cup towards the end of the execution. In the bottom row, for the *1 Hand 5 Fingers* embodiment, we observe that not completing the task and more specifically, leaving the pens in the first cup, generates a reward that is roughly half (image row 2, plot orange curve) the one achieved by a successful execution (image row 1, plot blue curve). These results are encouraging – they show that our learned encoder can represent fine-grained visual differences relevant to the task. Additionally, the training process for this visual reward did not require any additional environment instrumentation (apart from a camera), a desirable property for scaling to more complex, multi-step manipulation tasks.

Appendix D Additional Experimental Details

Our codebase is implemented in PyTorch [45]. Experiments were performed on a desktop machine with an AMD Ryzen 7 2700X CPU (8 Cores/16 Threads, 3.7GHz base clock), 32GB RAM, and a single NVIDIA GeForce RTX 2080 Ti GPU.

D.1 Representation Learning

Each representation learning run – specifically training and evaluating a representation and computing the final goal embedding vector – took an average of 25-30 minutes of wall clock time.

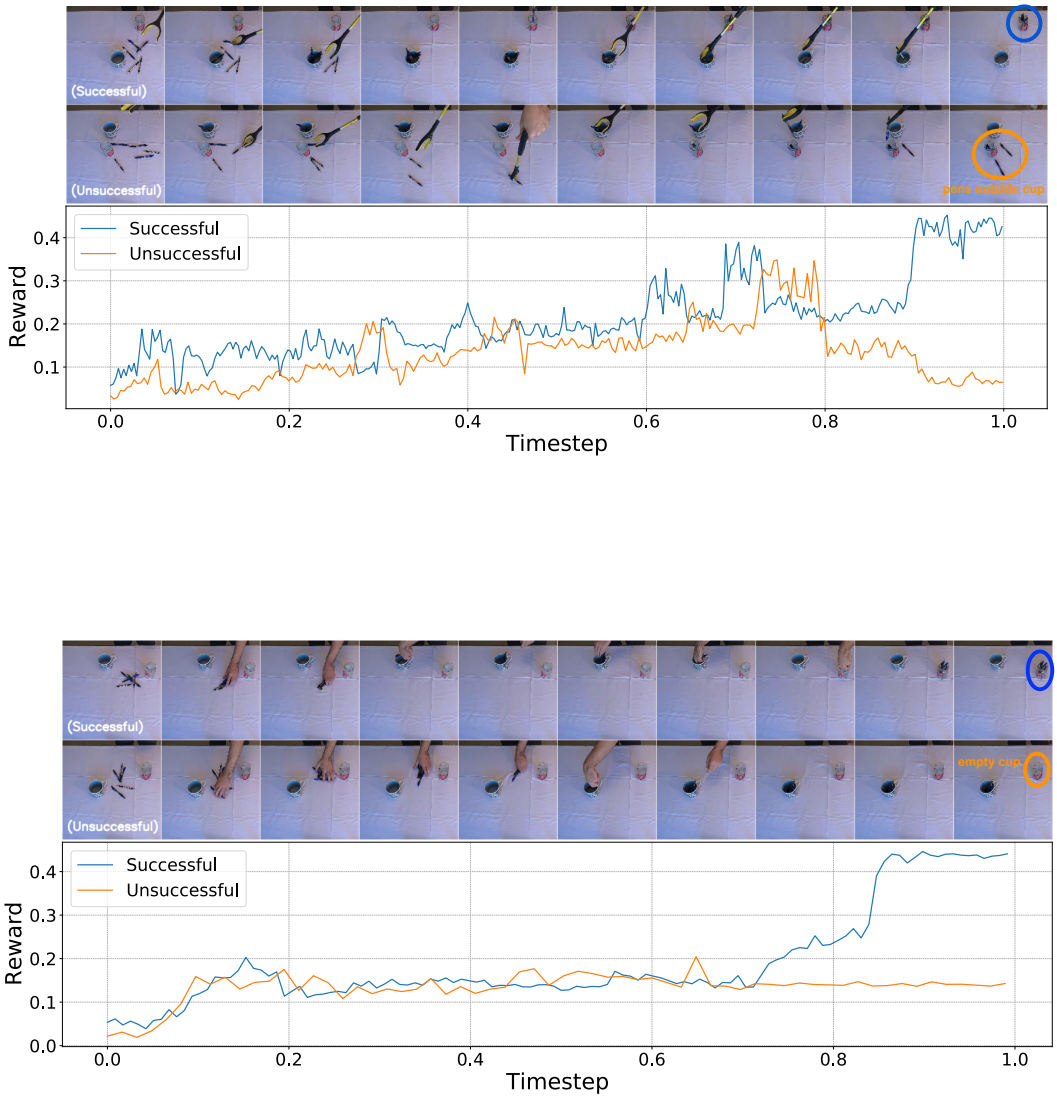

Figure 12. ***X-REAL "move pens to mug then cup" task:*** Visualizing our learned XIRL reward function on successful and unsuccessful demonstrations for the *RMS Grabber Reacher* (*top*) and *1 Hand 5 Fingers* (*bottom*) embodiments.

D.1.1 Additional Baseline Details

In this section, we provide a comprehensive overview of the baselines briefly described in Section 4.2.

ImageNet: We use an ImageNet pre-trained ResNet-18 with no additional training, i.e., we load the pre-trained weights, discard the classification head, and use the 512-dimensional embedding space from the penultimate layer.

Goal Classifier: A common strategy for using a learned visual reward function is to train a goal frame classifier [27] on a binary classification task where the last frame of all the demonstrations is considered positive and all the others are considered negatives. Since demonstration sequences do not

Parameter	Setting
Total train iterations	$6k$
Number of frames	15
Batch size	4
Embedding size	32
Unit normalize embeddings	False
Negative window size	5

Table 3. Hyperparameters used for the **Goal Classifier** baseline.

necessarily end exactly when the task is solved, we randomly sample the negatives from frames that are at least 5 indices away from the final frame. This was chosen heuristically by examining the dataset. Note that unlike [27], we do not have access to any unsuccessful trajectories in the dataset.

LIFS: We re-implement the approach from [16], which learns a feature space that is invariant to different embodiments using a loss function that encourages *corresponding* pairs of embodiment states across demonstrations to be close in the embedding space. We use the time-based alignment method described in their paper to find these corresponding pairs which assumes each embodiment performs the task at the same rate. To prevent the embeddings from collapsing to a constant value, they use an additional reconstruction loss to encourage the network decoders to preserve as much domain-invariant information as possible. We found early stopping to be crucial in preventing LIFS from collapsing to trivial embeddings. For this baseline, we also randomly sample N evenly-spaced frames from a video to construct each mini-batch.

Parameter	Setting
Total train iterations	$8k$
Number of frames	15
Batch size	4
Embedding size	32
Unit normalize embeddings	True
Embedding temperature	0.1

Table 4. Hyperparameters used for the **LIFS** baseline.

TCN: We re-implement the single-view variant of Time-Contrastive Network (TCN) [5] with positive and negative frame windows of 1 and 4 respectively. Different from [5], we do not use a time-indexed reward which is not applicable to agents with different embodiments. Like XIRL, we use the negative distance to the average goal embedding as the reward. For this baseline, we sample a contiguous chunk of N frames from a video to construct a mini-batch.

Parameter	Setting
Total train iterations	$4k$
Number of frames	20
Batch size	4
Embedding size	32
Unit normalize embeddings	True
Embedding temperature	learned
Positive window	1
Negative Window	4

Table 5. Hyperparameters used for the **TCN** baseline.

D.1.2 Data Augmentation & Preprocessing

We apply data augmentation during training using the **Albumentations** library [46]. Concretely, this involves the following transformations:

- `RandomResizedCrop`: This transformation takes a random crop from the original image, then resizes it to a final output height and width. The lower and upper bounds for the random crop area are set to $[0.6,1.0]$, the lower and upper bounds for the random aspect ratio of the crop are set to $[0.75,1.33]$ and the final output size is set to 224×224. We apply this transformation with a probability of 1.0 and denote it by C.
- `ColorJitter`: This transformation varies the brightness, contrast, saturation and hue of an input image. Our parameters for these respective changes are 0.4, 0.4, 0.1 and 0.1. We apply it with a probability of 0.8 and denote it by J.
- `ToGray`: This transformation converts an RGB image into a grayscale one. We apply it with a probability of 0.2 and denote it by G.
- `GaussianBlur`: This transformation blurs the input image with a Gaussian filter. We use a fixed kernel size of 13 and a standard deviation randomly sampled from the range $[1.0,2.0]$. We apply it with a probability of 0.2 and denote it by B.
- `Normalize`: Lastly, we divide the pixel values by 255 to scale their range to $[0,1]$. We apply it with a probability of 1.0 and denote it by N.

We apply the same randomly sampled transformations to all the sampled frames from the same video. However, independently sampled transformations are applied for each such frame stack in the mini-batch. Note that the order of transformation matters, i.e., we apply the following composed transform: $N\circ B\circ G\circ J\circ C$.

D.1.3 Training and Evaluation

All our representations are trained using an ADAM optimizer with $\beta_1=0.99$, $\beta_2=0.999$ and weight decay of 10^{-5}. While the representations are evaluated on the downstream policy learning performance, we also compute the following quantitative metrics and qualitative results on the train and validation sets to diagnose our representations:

- **Kendall's Tau**: A metric ranging from $[-1,1]$ that measures how well-aligned two sequences are in time. We refer the reader to [7] for a more in-depth explanation.

- **Nearest-Neighbor Alignment Video**: We randomly select one demonstration as a reference video. We use nearest-neighbor in the embedding space to align a test video with the reference video. See Appendix F and the supplemental video for example visualizations for both *same* and *cross*-embodiment settings. These videos highlight how well the embedding space encodes the task progress across different embodiments.

D.1.4 Other Hyperparameters

For a comprehensive list of hyperparameters used for representation learning on the X-MAGICAL environment, the Puck Pushing environment, and X-REAL (Appendix C), see Appendix G.1.

D.2 Policy Learning

The SAC [30] implementation we use is based off of [47]. Each run on the X-MAGICAL benchmark, i.e., the training and evaluation of a specific reward learning method on a specific embodiment with a single seed, took an average of 00h27m, 01h42m, 03h56m and 03h56m wall clock times for *long-stick*, *medium-stick*, *short-stick* and *gripper*, respectively. Note the difference in run times is due to the fact that each embodiment is trained for a different number of total training steps since each converges at different rates: 75k, 225k, 500k and 500k for *long-stick*, *medium-stick*, *short-stick* and *gripper* respectively. For the Puck Pushing experiments in Section 5.3, each run took an average wall clock time of 01h25m for 200k timesteps. Note that run times for both the above environments are recorded while performing up to 5 seed runs in parallel.

D.2.1 Soft-Actor Critic Architecture

We use clipped double Q-learning [48, 49] for the critic, where each Q-function is parameterized by a 3-layer multi-layer perceptron (MLP) with `ReLU` activations. The actor is implemented as a `tanh`-diagonal-Gaussian, and is also parameterized by a 3-layer MLP which outputs mean and covariance. Both actor and critic MLPs have a hidden size of 1024 – the weights are initialized with orthogonal [50] initialization, while the biases are initialized to zero.

D.3 Policy Input

As mentioned in Section 4.1 and Appendix B.1, we construct our observational inputs by stacking 3 consecutive state vectors. The input to the policy is thus a flattened vector in $\mathbb{R}^{48}$. For the *Puck Pushing* environment described in Section 5.3, we construct the observational input by stacking 3 consecutive state vectors containing the 3D Cartesian coordinates of the robot end-effector and the planar 2D coordinates of the puck. The input to the policy is thus a flattened vector in $\mathbb{R}^{15}$.

D.3.1 Training and Evaluation Setup

We first collect 5000 seed observations with a uniform random policy, after which we sample actions using the SAC policy. We then perform one gradient update every time we receive a new environment observation. When evaluating our agent every 5000 steps, we take the mean policy output (i.e., no sampling) and average the final success rate over 50 evaluation episodes.

D.4 Other Hyperparameters

For a comprehensive list of hyperparameters used for policy learning on the X-MAGICAL and Puck Pushing environments, see Appendix G.2.

Appendix E Additional Experiments

E.1 XIRL vs. Ground-truth Environment Reward

In this section, we compare the performance of our method XIRL against the ground-truth environment reward on the same-embodiment experiment from Section 5.1 and the cross-embodiment experiment from Section 5.2. We observe that in both same-embodiment (Figure 13, **top**) and cross-embodiment (Figure 13, **bottom**) settings across multiple embodiments, XIRL is either more or just as sample-efficient as the environment reward. This highlights XIRL's ability to provide denser reward information via encoding task progress, as opposed to the sparser ground-truth environment reward, which was shown in Figure 6(a).

E.2 XIRL vs. SimCLR

In this section, we compare the performance of XIRL against SimCLR [51], a constrative pretraining technique that has exhibited SOTA self-supervised performance on ImageNet. We implement two Sim-

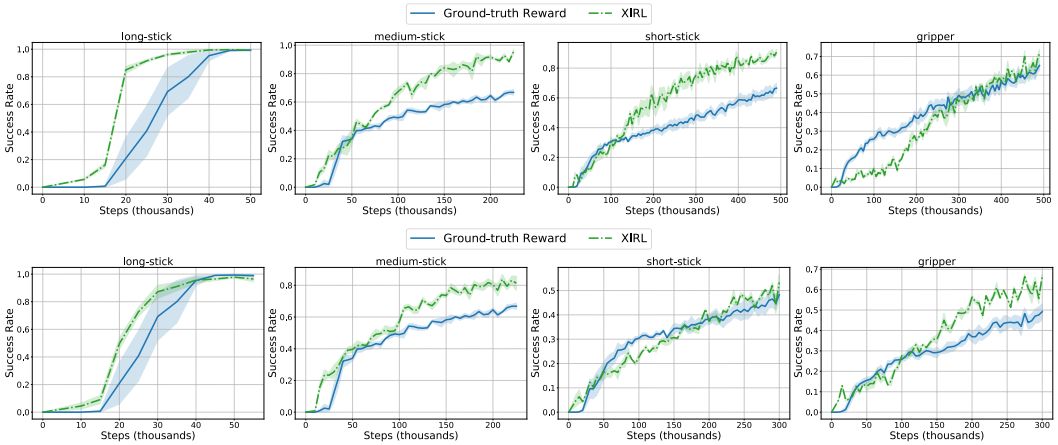

Figure 13. Comparison of XIRL with the ground-truth environment reward in the ***same-embodiment setting*** (top) and the ***cross-embodiment setting*** (bottom

CLR baselines: (a) *SimCLR* is a ResNet18 trained on the x-MAGICAL demonstration dataset with the constrastive pretraining pipeline described in [51], and (b) *SimCLR ImageNet* is a ResNet18 pretrained on ImageNet with SimCLR, with no further pretraining on x-MAGICAL. Below, we present results for the *longstick* and *mediumstick* embodiments of the x-MAGICAL benchmark:

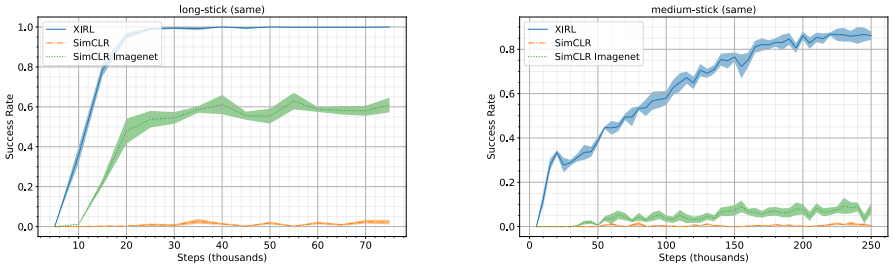

Figure 14. Comparison of XIRL & SimCLR in the ***same-embodiment setting***.

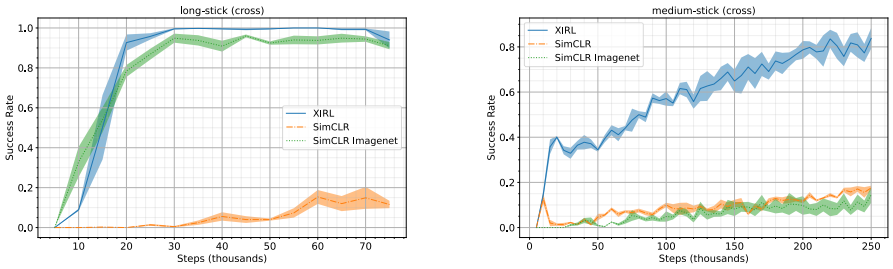

Figure 15. Comparison of XIRL & SimCLR in the ***cross-embodiment setting***.

We find that the SimCLR objective performs poorly when trained solely on the x-MAGICAL dataset, whereas the SimCLR baseline pertained on ImageNet (without finetuning on x-MAGICAL) does much better on the longstick embodiment (which is easier to solve). For the mediumstick embodiment, both perform poorly. XIRL, shown in blue, performs significantly better. This highlights that overall, both visual pretraining on cross-embodiment demonstrations and the inductive biases offered by the TCC loss are required to obtain good performance on downstream RL tasks.

Appendix F Qualitative Results

F.1 t-SNE Visualizations

To better understand and compare our learned representations, we visualize the t-SNE projection of the learned XIRL and Goal Classifier embedding spaces for 4 video demonstrations of the *shortstick* agent in Figure 16. We observe that:

- Trajectories for different demonstrations overlap and are well-aligned in the XIRL embedding space. In contrast, there is significantly less structure in the Goal Classifier space.
- Distances to the goal (top left corner in the top figure) correlate well to task progress.

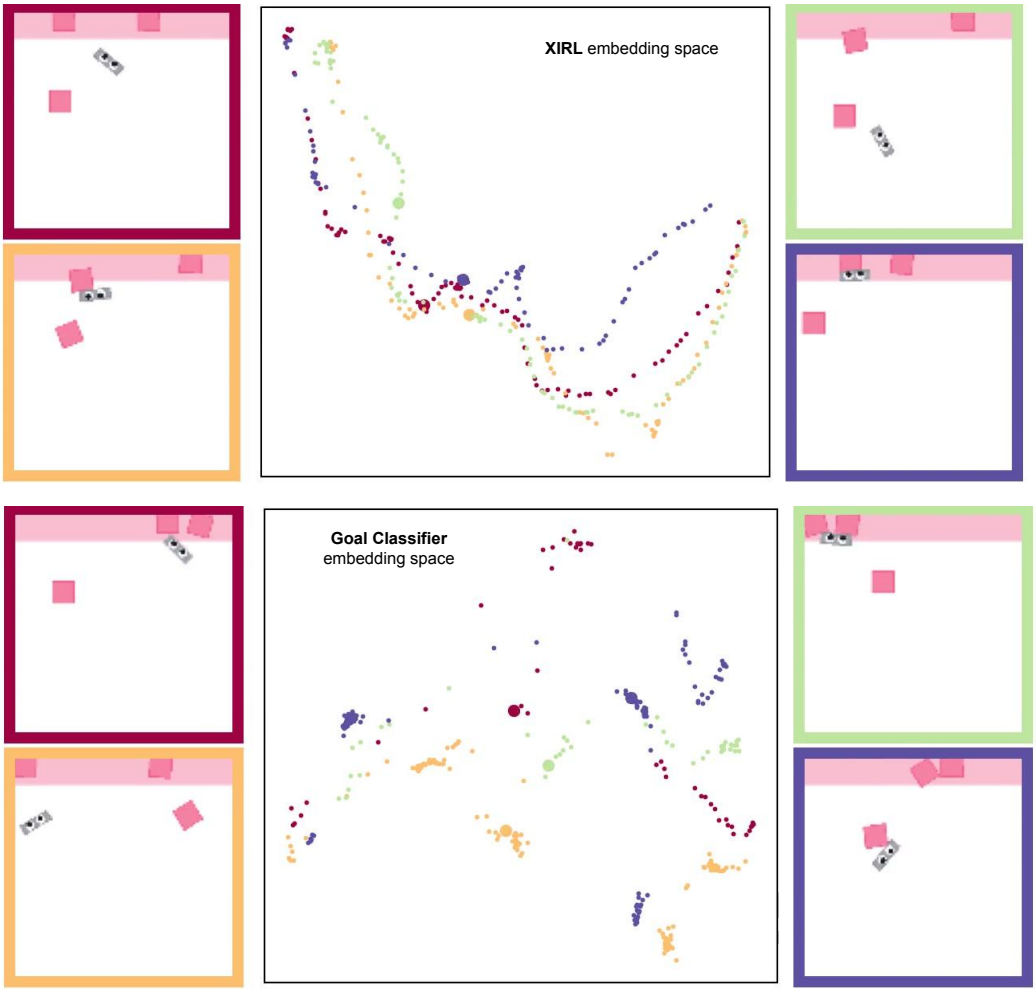

Figure 16. **t-SNE 2D projection** of the learned embedding space for XIRL (**top**) and Goal Classifier (**bottom**), where 4 video demonstrations are embedded and displayed for the *shortstick* agent from the X-MAGICAL benchmark.

We also provide video visualizations of the t-SNE embeddings in the supplementary video, which highlight some more properties.

F.2 Nearest-Neighbor Retrieval

We provide nearest-neighbor alignment videos in the supplementary video.

F.3 Other Visualizations

Please see the supplemental video for videos showing trained policy rollouts on the *Sweeping* and *Puck Pushing* tasks for the cross-embodiment setting, as well as interactive visualizations of the learned reward for the above environments and X-REAL.

Appendix G Hyperparameters

In this section, we give a comprehensive overview of the hyperparameters used for representation learning and policy learning on the *Sweeping* task from X-MAGICAL, the *Puck Pushing* task from [6], and X-REAL.

G.1 Representation Learning

We used mostly the same hyperparameters to train the XIRL encoder across all environments. The main parameters that vary are the embedding dimension and the number of sampled frames.

Hyperparameter	Value	Range Considered
Loss		
Type	regression (mse)	{mse, huber, x-ent}
Stochastic matching	False	-
Normalize time indices	True	-
Variance-aware	False	-
Distance metric	L2	{cosine, L2}
Optimizer		
Type	ADAM	-
Initial learning rate	10^{-6}	10^{-5} to 10^{-3}
Final learning rate	0	-
β_1	0.99	-
β_2	0.999	-
Weight decay	10^{-5}	10^{-6} to $5{\cdot}10^{-4}$
Misc.		
Total train iters.	$8{\cdot}10^3$	$2{\cdot}10^3$ to $30{\cdot}10^3$
Batch size	4	4 to 8
Normalize embeddings	False	{True, False}
Temperature	0.1	-
Learnable temperature	False	{True, False}
Augmentations	color jit, rand. crop, togray, gauss. blur	-
Frame Sampler	uniform	
Number of frames		
X-MAGICAL	40	5 to 50
Puck Pushing	40	5 to 50
X-REAL	30	5 to 50
Embedding size		
Sweeping	32	32 to 128
Puck Pushing	64	32 to 128
X-REAL	64	32 to 128

Table 6. Hyperparameters for all XIRL representation learning experiments.

G.2 Policy Learning

Most hyperparameters used for downstream reinforcement learning are identical across *X-MAGICAL* and *Puck Pushing*. What changes is the total number of training steps for each embodiment, since some converge much faster than others.

Hyperparameter	**Value**
Total train steps	
Gripper	500K
Shortstick	500K
Mediumstick	225K
Longstick	75K
Sawyer arm (RLV env)	200K
Optimizer	
Type	ADAM
Learning rate	10^{-4}
β_1	0.9
β_2	0.999
Q-network	
Hidden units	1024
Hidden layers	2
Non-linearity	`ReLU`
Actor	
Hidden units	1024
Hidden layers	2
Non-linearity	`ReLU`
Misc.	
Frames stacked	3
Action repetitions	1
Discount factor	0.99
Minibatch size	1024
Replay period every	1 step
Eval period every	5000 step
Number of eval episodes	50
Replay buffer capacity	10^6
Seed steps	5000
Critic target update frequency	2
Actor update frequency	2
Critic target EMA momentum (τ_Q)	0.005
Actor log std dev. bounds	$[-5,2]$
Entropy temperature	0.1
Learnable temperature	True

Table 7. Hyperparameters for all RL experiments.