A Appendix

A.1 Background on Pairwise Learning

We coin the term "pairwise" learning for the frameworks that learn visual representations based on semantic invariance between dual-stream encoder representations. A general pairwise learning method first generates multiple augmented views by applying a series of random image augmentations to the input sample, then clusters views with the same semantics in the representation space. Optionally in such frameworks, methods using contrastive losses repel samples with different semantics. Previous works not only have built various self-supervised tasks that benefit representation learning but also show that learned representations can benefit different downstream tasks.

In this paper, we focus on four representative pairwise learning methods, MoCo [13, 14, 34], BYOL [24], SimSiam [12] and DINO [8]. Specifically, MoCo takes advantage of the contrastive loss and negative samples in the mini-batch, while BYOL, SimSiam, and DINO focus on the similarity of the same image across diverse augmentations.

MoCo Momentum **Co**ntrast (MoCo) takes advantage of a contrastive loss function InfoNCE [57] with dot product similarity. It starts from two identical encoder networks, an online encoder f_q and a momentum encoder f_k.

At each training step, a mini-batch of N images x are uniformly sampled from a training set D. Given two distributions of image augmentations $\mathcal{T}$ and $\mathcal{T}'$, two image augmentations $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$ are sampled respectively and applied to x, resulting in $2N$ samples. Augmented images, $v = t(x)$ and $v' = t'(x)$, are called *views*. Then, v and v' are fed to two encoders to generate queries $q = f_q(v)$ and keys $k = f_k(v')$.

For each view v_i in v and its corresponding query $q_i = f_q(v_i)$, the contrastive loss is formulated as:

$$\mathcal{L}_{\text{MoCo},q_i} = -\log \frac{\text{sim}(q_i, k_i)}{\sum_{j=1}^{N} \text{sim}(q_i, k_j)} \tag{2}$$

where $\text{sim}(q_i, k_i) = \exp(q_i \cdot \text{sg}[k_i]/\tau)$, $\text{sg}[\cdot]$ implies stop gradients and τ is a temperature hyper-parameter. This loss encourages q_i to be similar to its corresponding key k_i (called *positive*), but dissimilar to other keys (called *negatives*) in the mini-batch. The online encoder f_q with parameters θ_q is updated by the above contrastive loss. The momentum encoder f_k with parameters θ_k, an Exponential Moving Average (EMA) of f_q, is updated by

$$\theta_k \coloneqq m\theta_k + (1-m)\theta_q, \tag{3}$$

where $m \in [0, 1)$ is a momentum coefficient that controls how fast θ_k updates towards the online network θ_q. Finally, f_k will be discarded once the training completes.

BYOL Similar to MoCo, in addition to f_q and f_k, BYOL maintains two identical projection networks g_q, g_k and one prediction networks p_q (See Fig. 10a). BYOL also starts from inputs v and v' but calculates the projection $z_1 = g_q(f_q(v))$ and $z_2 = g_k(f_k(v'))$, and tries to regress z_2 from z_1 using the prediction network p_q.

After applying $l2$-normalization to the prediction $p_q(z_1)$ and the target projection z_2, a mean squared error is measured as:

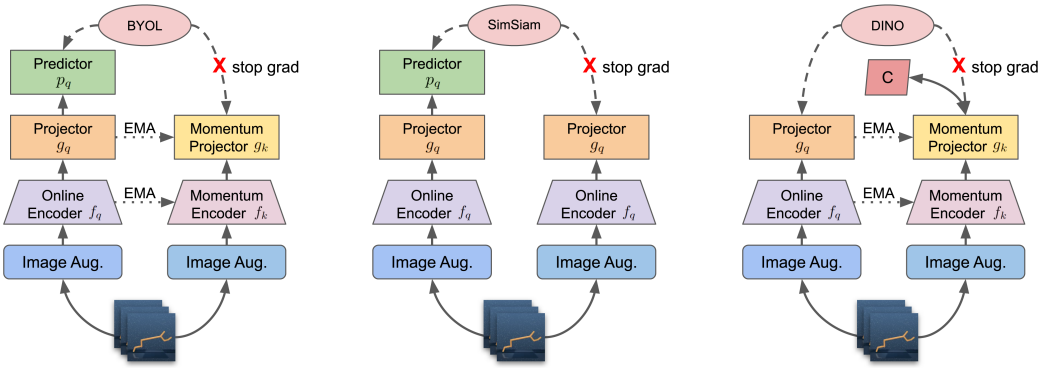
$$\mathcal{L}_{\text{BYOL},1} = \|p_q(z_1) - \text{sg}[z_2]\|_2^2 = 2 - 2\frac{p_q(z_1) \cdot \text{sg}[z_2]}{\|p_q(z_1)\|_2 \cdot \|\text{sg}[z_2]\|_2} \tag{4}$$

whose value is low when $p_q(z_1)$ is close to z_2.

Similarly, by swapping v and v', another symmetric loss can be applied on top of $z_1' = g_q(f_q(v'))$ and $z_2' = g_k(f_k(v))$, as $\mathcal{L}_{\text{BYOL},2} = \|p_q(z_1') - \text{sg}[z_2']\|_2^2$. The total loss is $\mathcal{L}_{\text{BYOL}} = (\mathcal{L}_{\text{BYOL},1} + \mathcal{L}_{\text{BYOL},2})/2$. The parameters of f_k and g_k are also the EMA of f_q and g_q respectively.

Finally, f_k, g_q, g_k and p_q will be discarded once the training completes.

SimSiam SimSiam (**Sim**ple **Siam**ese) shares the same architecture as BYOL, while the parameters of the 'momentum' branch of SimSiam are always tied to the 'online' branch (See Fig. 10b). Therefore, SimSiam only maintains one branch, including an encoder f, a projector g, and a predictor p.



(a) BYOL (b) SimSiam (c) DINO

Figure 10: Conceptual comparison of three pairwise learning frameworks

SimSiam uses negative cosine similarity to encourage the predicted representation $h = p(g(f(v))$ to be similar to the projected representation of another view $g(f(v'))$, as follows:

$$\mathcal{L}_{\text{SimSiam},1} = -\frac{h}{\|h\|_2} \cdot \frac{\text{sg}[g(f(v'))]}{\|\text{sg}[g(f(v'))]\|_2} \tag{5}$$

Another symmetric loss term can also be derived as $\mathcal{L}_{\text{SimSiam},2} = -\frac{h'}{\|h'\|_2} \cdot \frac{\text{sg}[g(f(v))]}{\|\text{sg}[g(f(v))]\|_2}$, where $h' = p(g(f(v')))$. The total loss is $\mathcal{L}_{\text{SimSiam}} = (\mathcal{L}_{\text{SimSiam},1} + \mathcal{L}_{\text{SimSiam},2})/2$. And g_q will be discarded once the model is trained.

DINO DINO shares a similar overall structure as MoCo which contains two encoders f_q and f_k, and f_k is the EMA of f_q (See Fig. 10c). The outputs of both encoder networks are normalized as probability distributions over K dimensions by applying softmax with a temperature parameter τ_t, and K is the dimension of $f_q(v)$. DINO also maintains a centering vector C with dimension K. Following the formulation of knowledge distillation, a cross-entropy loss is applied to encourage the output distribution of f_q to become similar to a centered distribution from f_k, as follows:

$$\mathcal{L}_{\text{DINO},1} = -P(\text{sg}[f_k(v')] - C) \cdot \log P(f_q(v)) \tag{6}$$

where $P(x) = \text{softmax}(x/\tau_t)$. By swapping v and v' in Eq. 6, another loss $\mathcal{L}_{\text{DINO},2}$ which is symmetric to $\mathcal{L}_{\text{DINO},1}$ can be derived. And the total loss is the mean value of $\mathcal{L}_{\text{DINO},1}$ and $\mathcal{L}_{\text{DINO},2}$.

After each step of optimization, f_k is updated by Eq. 3. C also gets updated in a similar manner:

$$C \coloneqq m_c C + (1 - m_c) \cdot \text{mean}(f_k(v), f_k(v')) \tag{7}$$

Here $m_c \in [0, 1)$ is another momentum coefficient.

A.2 Implementation Details

Here we present the implementation details in all settings.

A.2.1 General Joint Learning Framework

The general joint learning framework starts from the official implementation of CURL [45] for DMControl and Atari. For different self-supervised learning losses, we only replace the contrastive learning head of CURL with a different SSL head and update the loss calculation. All the hyper-parameters are left untouched, except that we use the learning rate 10^{-3} for all DMControl environments. The detailed hyper-parameters can be found at Table 2 (DMControl) and Table 4 (Atari). We keep the most of hyper-parameters from DMControl for real-world robot experiments. The modified configuration is listed as Table 3.

We use the official implementations of DrQ [83] and RAD [46] for DMControl benchmark. The re-implementation of DrQ (denoting as DrQ*) has the same joint learning framework and image augmentation from CURL.

Table 2: hyper-parameters used for DMControl with general joint learning framework

Hyperparameter	Value
Image augmentation	Random crop
Image size before augmentation	(100, 100)
Image size after augmentation	(84, 84)
Replay buffer size	100000
Number of environment step	100000
Initial explore steps	1000
Stacked frames	3
Action repeat	2 finger, spin; walker, walk; reacher, hard; hopper, hop 8 cartpole, swingup 4 otherwise
Critic target update frequency	2
Actor update freq	2
EMA τ for Q', g_k	0.01
EMA τ for f_k	0.05
Discount γ	.99
Initial α	0.1
Convolutional layers in f_q	4
Number of filters	32
Fully connected layer in f_q	1
Tanh after f_q	False
Image representation dimension	50
MLP layer of Q_q^i, A_p	3
MLP Hidden units	1024
MLP Non-linearity	ReLU
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_q, Q_q^i, A_p)$	(.9, .999)
$(\beta_1, \beta_2) \rightarrow (\alpha)$	(.5, .999)
Learning rate (f_q, Q_q^i, A_p)	10^{-3}
Learning rate (α)	10^{-4}
Batch size	512
Evaluation episodes	10
Train with random seeds	10

Table 3: Modified hyper-parameters for real-world robot experiments

Hyperparameter	Value
Stacked frames	1
Action repeat	1
Number of environment step	100000
Number of random seeds	10

A.2.2 Losses for Self-supervised Learning

Pairwise Learning In this section, we replace the contrastive learning head with projectors and encoders depending on the exact loss. All the projectors and predictors are two-layer MLPs with ReLU in the middle. The input dimension which is the output dimension of the encoder, (50 on DMControl and uArm Reacher, and 576 on Atari). The hidden dimension of the MLP is 256 and the output dimension is 128. We use the same encoder EMA update rate ($\tau = 0.05$ for SAC and 0.001 for Rainbow) to update the projectorg_k (if applicable) in the target branch. The applied losses are introduced in Sec. A.1.

Table 4: hyper-parameters used for Atari with general joint learning framework

Hyperparameter	Value
Image augmentation pipeline with image size	Original Image (84, 84)$\rightarrow$ Random crop (80, 80) $\rightarrow$ Replication padding (88, 88) $\rightarrow$ Random crop (84, 84)
Replay buffer size	100000
Number of environment step	400000
Initial explore steps	1600
Stacked frames	4
Frameskip	4
Action repeat	4
Discount γ	.99
Priority exponent	0.5
Priority correction	$0.4 \rightarrow 1$
Target update frequency	2000
Support of Q distribution	51 bins
EMA τ for f_k	0.05
Reward Clipping	$[-1, 1]$
Max gradient norm	10
Convolutional layers in f_q	2
Number of filters	(32, 64)
Image representation dimension	576
Fully connected layer type	Noisy Nets
Noisy nets parameter	0.1
MLP layer of Q_q	2
MLP Hidden units	256
MLP Non-linearity	ReLU
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_q, Q_q)$	(.9, .999)
Learning rate	10^{-4}
Batch size	32
Evaluation episodes	10
Train with random seeds	20

Transformation Awareness We use a two-layer MLP with ReLU as the classifier for both rotation classification and shuffle classification. The hidden dimension of the MLP is 1024 and the classifier is supervised by a cross-entropy loss. The output dimension is 4 for four-fold rotation classification and 1 for binary shuffle classification.

Reconstruction In this section, we follow the official implementation of SAC+AE [84] and apply the same image augmentation from CURL. The decoder has one fully connected layer and the same number of transposed convolutional layers as the convolutional layers in the encoder. When the output image from the decoder is smaller than the ground truth, we crop the ground truth to the size of the decoder output from the upper left corner.

For MAE we start from augmented SAC+AE and first divide the augmented image into non-overlapping patches in the spatial domain with a size of 4×4. Then we randomly mask 50% of the patches by setting the pixel value of the masked patches to zero. Finally, the reconstruction loss is modified to calculate MSE only over the masked patches. Other regularization losses are left untouched.

RL Context Prediction For all kinds of losses, the dimensions of all the fully connected layers and hidden layers in MLPs are 1024.

A.2.3 Manually Balance Two Self-supervised Losses

In this section, we further explore the ways to manually combine two self-supervised losses. Extract-AR, Guess-AF, and Predict-FR are methods manually designed to combine two individual losses However, Guess-AF and Predict-FR are not better than the single self-supervised loss in their combinations (see Guess-Action and Predict-Reward in Table 11 and Fig. 4). Considering that Extract-AR, Guess-AF, and Predict-FR concatenate both the outputs and apply supervision by averaging loss per element of the output, the target with a higher dimension will naturally get more penalty due to the larger number of elements in the output. We further test the 'Balanced' configuration, where we only modify how the supervision is applied. Take Extract-AR as an example, in the 'Balanced' setting, we first calculate loss regarding action prediction and reward prediction separately, then the total self-supervised loss is the average of both the action prediction loss and the reward prediction loss. By adjusting the combination weights, the 'Balanced' trick brings overall improvements on top of all three methods as shown in Table 5. Such observation suggests that we need to carefully design how the two losses are combined, which is getting trickier as the number of combined losses increases.

Table 5: Scores on DMControl improved by manually balancing two self-supervised losses, suggesting the importance of weight hyper-parameters when combining multiple losses. Methods in gray are without a self-supervised loss for reference.

Agent	ball_in_cup, catch	cartpole, swingup	cheetah, run	finger, spin	reacher, easy	walker, walk
SAC-Aug(100)	541.4 ± 306.2	563.4 ± 235.0	172.1 ± 64.0	724.6 ± 154.9	654.4 ± 222.1	422.1 ± 250.8
RAD	879.9 ± 82.0	786.4 ± 95.1	387.9 ± 81.3	910.4 ± 104.5	508.8 ± 111.5	522.1 ± 95.5
DrQ	914.9 ± 21.2	692.2 ± 222.9	360.4 ± 67.7	935.6 ± 201.3	713.7 ± 147.6	523.9 ± 182.2
Extract-AR	822.2 ± 240.5	592.9 ± 124.7	225.8 ± 60.7	783.0 ± 112.0	678.7 ± 181.3	458.4 ± 148.9
Extract-AR-Balanced	897.9 ± 113.9(↑75.7)	582.3 ± 119.2(↓-10.6)	232.9 ± 33.2(↑7.1)	881.5 ± 114.2(↑98.5)	720.5 ± 136.4(↑41.8)	533.8 ± 100.8(↑75.4)
Guess-AF	329.8 ± 298.4	140.7 ± 144.0	0.9 ± 22.8	880.0 ± 59.5	382.9 ± 265.0	494.7 ± 112.7
Guess-AF-Balanced	918.4 ± 353.5(↑588.6)	536.1 ± 190.3(↑395.4)	191.2 ± 78.6(↑190.3)	842.9 ± 67.9(↓-37.1)	462.0 ± 208.7(↑79.1)	507.0 ± 128.7(↑12.3)
Predict-FR	750.3 ± 256.0	723.2 ± 167.5	12.4 ± 35.7	861.5 ± 49.2	636.1 ± 201.4	270.0 ± 154.9
Predict-FR-Balanced	829.6 ± 241.6(↑79.3)	751.0 ± 90.0(↑27.8)	216.2 ± 77.6(↑203.8)	864.8 ± 72.2(↑3.3)	882.1 ± 87.4(↑246.0)	472.9 ± 189.7(↑202.9)

A.2.4 Evolving Multiple Self-supervised Losses

We choose PSO (**P**article **S**warm **O**ptimization) [41] for the optimal combination of hyper-parameters including N_w weights of losses $w_{i=1,2,...,N_w}$ and two magnitudes of augmentation $m_{j=1,2}$ for the online networks and the target networks respectively. Each m_j varies from $[84, 116]$. We limit the range of each w_i to $[0, 10]$ for ELo-SAC and ELo-Rainbow while a range of $[10^{-4}, 10^4]$ for ELo-SACv2 and ELo-SACv3.

During the evolutionary search, we use a batch size of 128 for ELo-SAC and ELo-SACv2, each combination is trained with 5 different random seeds. As for ELo-SACv3, the batch size is set to 64 and the number of random seeds is set to 3 to save computation. ELo-Rainbow also train with 5 random seeds during the evolutionary search. Other hyperparameters used in the search and all hyperparameters for evaluation are identical to Table 2 and Table 4.

ELo-SAC ELo-SAC maintains a population of 50 for DMControl and each particle evolves 15 generations in "cheetah, run". Before the search, the first i^{th} particles are initialized with $m_{j=1,2} = 88$, and each particle only has one weight set to 1 and other weights set to 0. In another word, these first i^{th} particles start with the existing single self-supervised loss method in the search space. Other particles are randomly initialized. Table 6 shows the combination Elo-SAC found in cheetah run. The columns in Table 6 show the search space. The first six columns denote the optimal weight w_i of its corresponding loss obtained with the evolutionary search, while the last two columns denote the original image size before random crop (image augmentation magnitude $m_{j=1,2}$).

Table 6: Optimal parameters that ELo-SAC found in cheetah run

Agent	Searched Env.	CURL w_1	BYOL w_2	Predict FR w_3	Extract AR w_4	AutoEncoder w_5	RotationCLS w_6	Online Aug. m_1	Target Aug. m_2
ELo-SAC	Cheetah, run	0	0.288	0.628	0	0	0.009	87	86

ELo-SACv2 Compared with ELo-SAC, ELo-SACv2 has the following major improvements:

1. **Initialization** Define the set of image augmentation magnitude $\mathcal{M} = \{(m_1 = t, m_2 = t) \mid t = 86, 88, 92, 100, 116\}$, and the set of SSL weights $\mathcal{W} = \{(w_{i=t} = 1, w_{i\neq t} = 0) \mid t = 1, 2, ..., 6\}$. The first $|\mathcal{M}| \times |\mathcal{W}|$ particles are initialized from the Cartesian product of $\mathcal{M}$ and $\mathcal{W}$. Other particles are randomly initialized.
2. **Search space** The search space of self-supervised losses is updated based on the loss performance at Table 11. We empirically choose the losses from different categories that have a relatively strong performance when it is applied solely to RL.
3. **Weight range** The weight of each self-supervised loss is presented on a log scale so that the search can cover a larger range.

Besides the improvements above, ELo-SACv2 evolves 45 generations and the optimal combination is chosen from the top 10 combinations regarding the overall performance. The optimal combination found by ELo-SACv2 is shown in Table 7. ELo-SACv2 slightly improves the results of ELo-SAC with all the modifications (see Figure 4 and Table 11).

Table 7: Optimal parameters that ELo-SACv2 found in cheetah run

Agent	Searched Env.	CURL $\log_{10} w_1$	DINO $\log_{10} w_2$	Predict FR Balanced $\log_{10} w_3$	Extract AR Balanced $\log_{10} w_4$	AutoEncoder $\log_{10} w_5$	RotationCLS $\log_{10} w_6$	Online Aug. m_1	Target Aug. m_2
ELo-SACv2	Cheetah, run	-3.309	-0.562	1.272	-0.772	-3.904	0.344	88	91

ELo-SACv3 Since ELo-SAC and ELo-SACv2 only search in one DMControl environment, 'cheetah run', and both the found solutions perform weaker on 'finger, spin' and 'reacher, easy', we further extend ELo-SACv2 to search in multiple environments at the same time, named ELo-SACv3. The optimization process of ELo-SACv3 is presented as:

$$\underset{m_{j=1,2}, w_{i=1,\dots,N_l}}{\operatorname{argmax}} \quad \text{mean}(\hat{\mathcal{R}}_{\text{envs}}^{\text{seed=1,2,3}}(m_{j=1,2}, w_{i=1,\dots,N_l})) \tag{8}$$

where $\hat{R} = R/R_{\text{DrQ}}$ is the original agent reward R normalized by the score of DrQ R_{DrQ} reported in [83], and envs is the set of six DMControl environments listed in Table 11.

We let ELo-SACv3 evolve for 25 generations and chose the loss combination with best performance among the top 10 records. The found parameters are listed in Table 8.

Table 8: Optimal parameters that ELo-SACv3 found in six DMControl environments

Agent	Searched Env.	CURL $\log_{10} w_1$	DINO $\log_{10} w_2$	Predict FR Balanced $\log_{10} w_3$	Extract AR Balanced $\log_{10} w_4$	AutoEncoder $\log_{10} w_5$	RotationCLS $\log_{10} w_6$	Online Aug. m_1	Target Aug. m_2
ELo-SACv3	6 environments	-2.304	-4.0	-2.989	0.103	-1.722	-3.481	88	89

ELo-Rainbow ELo-Rainbow has a population of 30 and the initialization is similar to ELo-SAC. The search is performed on Frostbite only for 10 generations and the found combination is shown in Table 9.

Table 9: Parameters of ELo-Rainbow found in Frostbite

Agent	Searched Env.	BYOL w_1	Predict Future w_2	Extract Reward w_3	AutoEncoder w_4	Rotation CLS w_5
ELo-Rainbow	Frostbite	0.250	1.054	2.280	0.953	0.591

Interestingly, we find that the optimal combination found by ELo-SAC is relatively sparse, where BYOL and Predict FR are the only two major losses. Similarly, ELo-SACv2 relies more on Predict-FR-Balanced and RotationCLS, while ELo-SACv3 relies on Extract-AR-Balanced mostly. However,

for ELo-Rainbow, the magnitudes of all the weights are relatively similar. The difference between the found results reflects the different properties of different environments. Our further experiments in DMControl confirm the generalization ability to evolve losses; i.e., the obtained solution of weights in one environment achieves relatively good performance on other environments in the same benchmark. However, results on Atari are much inconsistent with DMControl. We cover detailed observations and discussions in Section 4 and Appendix A.4.

The code for ELo-SACv3 is available at `https://github.com/LostXine/elo-sac`, and the code for ELo-Rainbow is available at `https://github.com/LostXine/elo-rainbow`.

A.2.5 Comparison of method variants

In Section 4, several variants of the existing methods are introduced. The difference between these methods, especially on image augmentation, can be summarized as follows: SAC-NoAug is the original pixel-based SAC [27, 28]. SAC-Aug(88) and SAC-Aug(100) use the random crop as the only image augmentation, where (88) means the original image has a size of 88×88 before randomly cropping to 84×84 and (100) means the original image has a size of 100×100. These two methods should be regarded as variants of RAD with different augmentation choices. The random crop from 100×100 to 84×84 is the default image augmentation method for all the methods introduced in Sec. 3.2, including SAC+AE. Essentially, if we remove their self-supervised loss, they will fall back to SAC-Aug(100). Similarly, we test DrQ variants by replacing its default random shift augmentation with the random crop, reported as DrQ(88) and DrQ(100). Meanwhile, RAD uses random translate by default except on walker walk; ELo-SAC and ELo-SACv2 first crop the center of the input image to the found optimal sizes. Then two central patches are randomly cropped to 84×84 as the inputs for the online networks and the target networks respectively.

For the policy learning part, all the methods share the same model. However, DrQ, DrQ(88), DrQ(100), and SAC+AE apply an additional $\tanh$ activation after the visual encoder. We also study the effect of the activation function in the coming Section A.3.4.

A.3 Ablations

Besides random crop and encoder backbone investigated in Section 4, we further perform detailed ablations on more image augmentation, learning rate, encoder architecture, and activation function in this section. The default test environment is identical to ablations in Section 4.

A.3.1 Image Augmentation

We study the effect of random translate, an image augmentation method which is widely used in RAD [46]. Similar to random crop, the image size for translate is linear to the magnitude of the translate, when using a fixed crop size: the larger the image size, the stronger the augmentation. As shown in Fig. 11, image size for translate has a similar pattern for most tested methods (with an exception of RotationCLS). In summary, it is critical to engineering image augmentation carefully when designing an RL system with or without SSL.

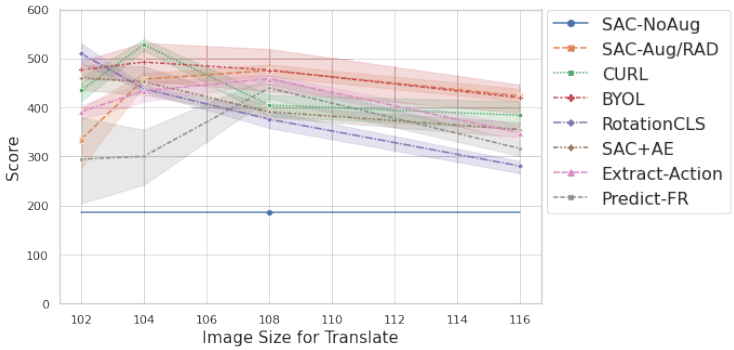


Figure 11: Ablations on translate image augmentation.

A.3.2 Learning Rate

Fig. 12 shows how the learning rate of self-supervised loss contributes to the performance. In this group of ablations, we only change the learning rate for SSL and leave the RL part untouched. SAC-NoAug and SAC-Aug(100) are both baselines for reference without any self-supervised losses. The results suggest that a smaller learning rate for SSL may improve performance. Therefore, it is necessary to search for the absolute weights of losses like ELo [60], which is equivalent to searching for the learning rate.

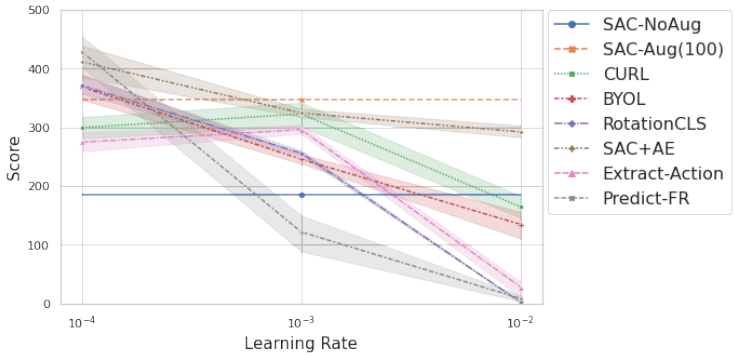


Figure 12: Ablations on the self-supervised learning rate.

A.3.3 Encoder Architecture

We further investigate the effect of additional linear layers in the visual encoder. Additional linear layers with ReLU activation are appended to the end of the visual encoder. All additional layers have a latent dimension of 128. Fig. 13 shows that additional layers usually bring downgraded performance, which could be caused by limited data.

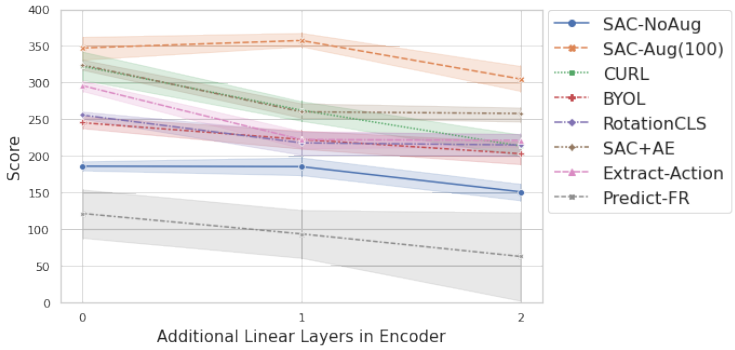


Figure 13: Ablations on additional linear layers after the visual encoder.

Another important aspect of the encoder architecture design is how to merge or separate two branches. As Visionary [3] suggests, where and how to merge visual representation with action representation is critical when designing an efficient value network. Similarly, we hypothesize that the point where split the representation for the SSL branch and the RL branch is also important. Figure 14 demonstrates two separation point configurations, named A and B. Figure 15 shows how the performance of different approaches changes in such two configurations.

A.3.4 Activation Function

Though all the methods we tested in Table 11 share the same visual encoder architecture, DrQ and SAC+AE apply an additional activation function $\tanh$ to the visual representation. To make a fair comparison and study the effect of such an activation function, we conducted detailed ablations on

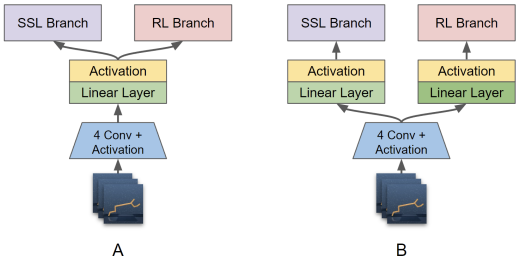


Figure 14: Comparison of two separation points.

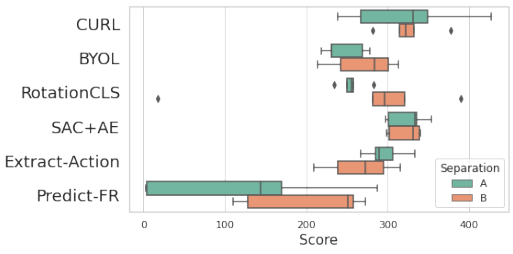


Figure 15: Ablation on separation points.

six DMControl tasks using the hyper-parameters identical to Table 2. Results in Table 10 confirm that the default design choice of all three methods is better than their alternatives.

Table 10: Ablation on Tanh activation

Agent	$Tanh$	ball_in_cup, catch	cartpole, swingup	cheetah, run	finger, spin	reacher, easy	walker, walk	Relative Score
DrQ (default)	✓	**914.9 ± 21.2**	692.2 ± 222.9	360.4 ± 67.7	**935.6 ± 201.3**	**713.7 ± 147.6**	523.9 ± 182.2	**6.642**
DrQ-w/o-Tanh		870.8 ± 177.0	**826.4 ± 44.9**	**393.7 ± 74.0**	849.6 ± 140.9	635.0 ± 155.0	**525.0 ± 163.7**	6.182
CURL-w-Tanh	✓	832.4 ± 118.4	508.5 ± 133.9	209.3 ± 24.2	676.3 ± 185.2	336.1 ± 216.5	463.7 ± 93.4	-3.266
CURL (default)		730.0 ± 179.4	471.5 ± 89.9	215.1 ± 57.3	717.8 ± 136.5	569.8 ± 179.4	442.6 ± 87.1	-2.032
SAC+AE (default)	✓	616.1 ± 169.9	388.8 ± 130.1	291.8 ± 59.8	799.0 ± 138.9	481.3 ± 130.4	402.6 ± 161.5	-2.529
SAC+AE-w/o-Tanh		358.9 ± 209.8	378.0 ± 96.0	289.4 ± 60.7	702.5 ± 157.3	516.8 ± 190.0	375.4 ± 136.2	-4.998

A.4 Detailed Results on DMControl and Atari

DMControl Table 11 notes the Interquartile mean, standard deviation and Relative Scores of tested algorithms in six DMControl environments. The score distribution of tested algorithms over six environments is summarized as Figure 20. Table 12 and Figure 21 include results of two additional harder environments in DMControl. The figures of reward curve v.s. environment step are grouped as Figure 23-32 and Figure 33-37 by the learning method.

Atari Similar to Table 11 and Figure 20, Table 13 Figure 22 are results in seven Atari environments.

A.5 Real-world Robot Experiments

To further evaluate methods in the real world applications, we set up a continuous robot arm control environment, uArm reacher. With the help of some simple techniques in computer vision and robotics, our environment can autonomously randomly reset and keep the agent training without any human input.

The environment requires a robotic arm with a suction cup actuator, two fixed RGB cameras, and a cube that can be picked up by the suction cup as the target, as shown in Fig. 8. The goal is to move the actuator close to the target as fast as possible. The observation comes from two cameras with a native resolution of 640×480. The images are then resized to 100×100, stacked along channel axis, and finally randomly cropped into 84×84, resulting in an $84 \times 84 \times (3 + 3)$ image observation before fed to the network. The action space is a 3D vector ranging from -1 to 1, and it will be mapped to the actuator position movement in a 3D robot Cartesian coordinates whose original point is the center of the robot base.The robot's motion range is manually limited for safety concerns while avoiding the actuator moving the target in one episode. Following reacher in DMControl, we define a very simple reward function. The reward function returns 1 when the 3D Euclidean distance between the actuator and the target is lower than a threshold, otherwise, it returns $-1e-3$. The length of each episode is set to 200 steps, which limits the range of the episode accumulated reward to $[-0.2, 200]$.

To enable automatic reward generation, we make an automatic calibration framework to get the target location in 3D, and calibrate the top-down camera before any experiments. We use AprilTag [56] to locate the robot position in the image plane, and read the 3D robot coordinates directly from the robot. By doing so, we can build a map between 2D image coordinates and 3D robot coordinates. The 2D coordinates of the target is first extracted by a simple color threshold. Then, given the constant

Table 11: Interquartile mean and standard deviation on six DMControl tasks. The last column is colored based on the relative performance w.r.t. SAC-Aug(100), Fig. 4

	Agent	ball_in_cup, catch	cartpole, swingup	cheetah, run	finger, spin	reacher, easy	walker, walk	Relative Score
No SSL	**SAC-NoAug**	71.4 ± 139.9	224.8 ± 28.6	120.9 ± 25.7	238.9 ± 172.6	204.8 ± 131.8	99.6 ± 38.7	-8.868
	SAC-Aug(88)	510.8 ± 187.4	714.2 ± 113.9	354.5 ± 68.7	771.2 ± 175.0	347.9 ± 148.5	192.2 ± 165.0	0.160
	SAC-Aug(100)	541.4 ± 306.2	563.4 ± 235.0	172.1 ± 64.0	724.6 ± 154.9	654.4 ± 222.1	422.1 ± 250.8	0.986
	RAD	879.9 ± 82.0	786.4 ± 95.1	387.9 ± 81.3	910.4 ± 104.5	508.8 ± 111.5	522.1 ± 95.5	5.310
	DrQ	914.9 ± 21.2	692.2 ± 222.9	360.4 ± 67.7	935.6 ± 201.3	713.7 ± 147.6	523.9 ± 182.2	**6.028**
	DrQ(88)	762.5 ± 139.4	508.2 ± 161.2	331.7 ± 80.5	877.6 ± 93.2	395.5 ± 161.0	119.2 ± 160.5	0.154
	DrQ(100)	907.6 ± 102.9	675.5 ± 131.1	318.8 ± 54.2	**940.0 ± 127.2**	627.0 ± 233.0	302.9 ± 295.8	3.898
Self-supervised	**CURL**	730.0 ± 179.4	471.5 ± 89.9	215.1 ± 57.3	717.8 ± 136.5	569.8 ± 179.4	442.6 ± 87.1	1.128
	CURL-w-Action	888.4 ± 179.5	537.8 ± 189.9	247.7 ± 72.7	604.2 ± 79.3	521.3 ± 211.0	439.9 ± 67.8	1.452
	CURL-w-Critic	690.9 ± 328.8	603.8 ± 156.4	233.7 ± 44.3	657.0 ± 127.1	536.3 ± 208.8	443.0 ± 157.9	1.320
	BYOL	667.7 ± 281.2	507.2 ± 221.7	70.7 ± 44.3	547.3 ± 185.6	403.7 ± 183.7	449.0 ± 153.5	-1.594
	DINO	**916.9 ± 65.7**	686.0 ± 152.2	198.3 ± 79.3	923.1 ± 124.4	686.2 ± 198.2	414.6 ± 162.4	3.957
	SimSiam	82.6 ± 86.7	67.4 ± 68.6	0.7 ± 0.3	7.6 ± 179.4	72.3 ± 71.1	34.1 ± 24.0	-12.537
	RotationCLS	157.9 ± 212.1	336.4 ± 220.1	209.7 ± 44.7	801.9 ± 139.7	540.3 ± 163.7	537.0 ± 170.3	-0.718
	ShuffleCLS	112.2 ± 101.9	28.8 ± 28.4	0.9 ± 0.4	53.0 ± 162.8	108.3 ± 55.4	127.3 ± 98.9	-11.701
	SAC+AE	616.1 ± 169.9	388.8 ± 130.1	291.8 ± 59.8	799.0 ± 138.9	481.3 ± 130.4	402.6 ± 161.5	0.566
	MAE	251.1 ± 231.1	372.8 ± 76.1	282.0 ± 62.3	669.5 ± 112.8	336.9 ± 170.1	489.7 ± 49.4	-1.635
	Extract-Action	871.0 ± 298.6	493.9 ± 162.7	172.3 ± 65.5	870.4 ± 108.1	578.3 ± 144.4	484.8 ± 70.5	2.297
	Extract-Reward	598.2 ± 306.2	469.8 ± 218.7	302.1 ± 89.9	828.7 ± 115.3	753.2 ± 155.5	522.2 ± 130.5	3.266
	Guess-Action	724.6 ± 265.3	495.7 ± 121.4	204.6 ± 26.2	669.9 ± 116.8	578.8 ± 161.1	410.6 ± 91.1	0.813
	Guess-Future	82.4 ± 87.1	146.6 ± 178.0	0.7 ± 0.4	786.5 ± 117.8	323.4 ± 229.2	74.1 ± 73.6	-7.318
	Predict-Future	121.5 ± 186.9	252.7 ± 219.9	0.7 ± 0.3	796.7 ± 166.7	365.3 ± 235.2	112.7 ± 137.4	-6.201
	Predict-Reward	672.8 ± 260.3	517.8 ± 215.6	279.1 ± 71.9	837.6 ± 264.6	**796.2 ± 143.5**	520.1 ± 218.1	3.826
	Extract-AR	822.2 ± 240.5	592.9 ± 124.7	225.8 ± 60.7	783.0 ± 112.0	678.7 ± 181.3	458.4 ± 148.9	3.042
	Guess-AF	329.8 ± 298.4	140.7 ± 144.0	0.9 ± 22.8	880.0 ± 59.5	382.9 ± 265.0	494.7 ± 112.7	-3.421
	Predict-FR	750.3 ± 256.0	723.2 ± 167.5	12.4 ± 35.7	861.5 ± 49.2	636.1 ± 201.4	270.0 ± 154.9	0.821
	ELo-SAC	831.3 ± 76.2	**798.7 ± 44.4**	354.0 ± 68.9	835.7 ± 151.2	485.2 ± 171.5	532.1 ± 160.7	4.567
	ELo-SACv2	864.6 ± 97.0	679.8 ± 104.7	**414.0 ± 59.8**	844.0 ± 166.4	513.9 ± 95.5	555.4 ± 163.7	4.901
	ELo-SACv3	851.0 ± 143.5	612.6 ± 87.7	313.9 ± 74.6	914.7 ± 143.4	625.2 ± 94.5	**697.4 ± 238.1**	5.502

Table 12: Scores on two harder tasks in DMControl

	Agent	hopper, hop	reacher, hard	Relative Score
No SSL	**SAC-NoAug**	0.033 ± 0.4	3.1 ± 39.7	-2.543
	SAC-Aug(88)	0.048 ± 0.4	210.733 ± 190.9	0.191
	SAC-Aug(100)	0.024 ± 0.4	**262.4 ± 140.4**	0.634
	RAD	0.038 ± 0.9	193.1 ± 186.1	-0.112
	DrQ	**0.424 ± 1.4**	258.95 ± 205.6	**4.017**
Self-sup.	**CURL**	0.076 ± 0.4	115.5 ± 116.4	-0.765
	BYOL	0.025 ± 0.1	49.725 ± 126.1	-2.025
	DINO	0.25 ± 0.5	200.333 ± 178.9	1.789
	RotationCLS	0.031 ± 0.1	210.05 ± 117.3	0.036
	SAC+AE	0.061 ± 0.4	140.567 ± 185.1	-0.579
	ELo-SAC	0.116 ± 0.3	81.592 ± 53.7	-0.845
	ELo-SACv2	0.147 ± 0.6	152.858 ± 70.8	0.314
	ELo-SACv3	0.177 ± 0.4	98.208 ± 78.7	-0.112

height of the target, we can obtain 3D target location from its 2D image coordinates according to the 2D↔3D map.

The environmental *reset* process is also automatic. At the beginning of each episode, the robot arm will pick up the target cube, and randomly release the cube at a certain height like throwing dice, in order to randomly initialize the cube location. The new location of the target cube is saved for generating rewards. After the robot arm automatically moves to a fixed pre-assigned starting point, the environmental reset is done and then the RL agent takes over the control. The RL agent can perform regular online training until the episode ends. Finally, after each episode ends, the environment repeats the reset process to initialize the next episode.

Table 13: Scores on Atari, the last column is colored based on the relative performance w.r.t. Efficient Rainbow, "*" means using a different image augmentation method from the original paper

	Agent	assault	battle_zone	demon_attack	frostbite	jamesbond	kangaroo	pong	Relative Score
No SSL	**Eff.-Rainbow**	506.8 ± 59.3	**14840.0 ± 6681.7**	519.3 ± 193.1	873.1 ± 834.8	318.5 ± 92.7	853.0 ± 1304.8	-19.0 ± 2.4	2.065
	Rainbow-Aug	459.7 ± 79.6	4770.0 ± 4379.0	870.3 ± 345.9	1469.7 ± 962.2	317.0 ± 110.5	619.0 ± 298.0	-20.3 ± 0.5	-2.223
	DrQ*	503.7 ± 89.0	7600.0 ± 6839.0	**891.2 ± 322.3**	943.7 ± 913.2	**321.0 ± 91.6**	605.0 ± 462.0	-19.9 ± 0.8	-0.290
Self-supervised	**CURL**	511.6 ± 107.3	5100.0 ± 5530.2	615.3 ± 240.4	928.3 ± 1018.5	307.0 ± 219.8	620.0 ± 300.8	**-18.1 ± 2.3**	-0.516
	BYOL	**514.6 ± 93.4**	9470.0 ± 4879.6	418.4 ± 246.5	**2111.5 ± 982.6**	291.5 ± 90.9	740.0 ± 1573.6	-18.5 ± 2.9	1.877
	RotationCLS	427.1 ± 62.2	12950.0 ± 5742.7	401.0 ± 159.0	1591.9 ± 949.5	285.5 ± 70.2	892.0 ± 1674.2	-19.3 ± 1.3	-2.647
	Rainbow+AE	485.2 ± 74.7	14290.0 ± 5927.7	528.8 ± 158.6	1272.5 ± 964.3	320.5 ± 68.8	**1155.0 ± 1392.5**	-18.8 ± 2.3	**4.815**
	Extract-Action	443.6 ± 72.8	7370.0 ± 3797.5	521.0 ± 126.6	1627.4 ± 874.5	282.0 ± 56.1	855.0 ± 612.3	-18.7 ± 2.5	-2.237
	Extract-Reward	494.8 ± 63.7	14420.0 ± 4901.0	533.4 ± 224.1	1286.6 ± 1109.0	294.5 ± 83.7	804.0 ± 1001.0	-18.4 ± 2.1	1.692
	Predict-Future	509.5 ± 67.7	10420.0 ± 5252.4	452.1 ± 145.6	1144.5 ± 988.7	295.0 ± 70.7	733.0 ± 966.0	-19.4 ± 2.1	-1.796
	Predict-Reward	485.6 ± 100.8	11870.0 ± 4197.2	547.9 ± 291.6	1155.9 ± 946.9	304.0 ± 92.7	908.0 ± 1718.9	-19.4 ± 1.7	0.135
	Predict-FR-Balanced	485.7 ± 82.2	14270.0 ± 4421.5	495.8 ± 209.7	1359.1 ± 1029.2	293.5 ± 146.8	664.0 ± 1239.6	-18.9 ± 1.3	-0.508
	ELo-Rainbow	493.1 ± 67.4	11750.0 ± 4727.9	623.4 ± 249.9	1027.6 ± 863.8	297.5 ± 66.4	795.0 ± 593.3	-19.2 ± 2.3	-0.369

A.6 Empirical Analysis on the Learned Representations

To further understand the role of self-supervised loss and image augmentation in an online reinforcement learning system with the joint learning framework, we empirically show the properties of representations learned by different losses.

We first follow Wang et al. [77] and measure the three metrics Dynamic Awareness, Diversity, and Orthogonality, extending them from discrete action space to continuous action space.

Dynamics Awareness means two states that are adjacent in time should have similar representations and states further apart should have a low similarity.

Diversity measures a ratio between state and state-value differences. High diversity means two states have two different representations to be distinguished even when they have similar state values.

Orthogonality reflects the linear independence of the representation, in another word, the higher the orthogonality, the lower the redundancy in the representations.

Assume an image observation x_i is taken when the intrinsic system state is s_i. Denoting the visual representation of x_i generated by the encoder from the critic networks as ϕ_i, and $\text{Critic}(\phi_i, \cdot)$ is the learned critic network output. Eq. 9 shows how to compute the three representation metrics.

$$
\begin{aligned}
\text{Dynamic Awareness} &= \frac{\sum_i^N \left\|\phi_i - \phi_{j\sim U(1,N)}\right\|_2 - \sum_i^N \|\phi_i - \phi_i'\|_2}{\sum_i^N \left\|\phi_i - \phi_{j\sim U(1,N)}\right\|_2} \\
\text{Diversity} &= 1 - \frac{1}{N^2}\sum_{i,j}^{N} \min\left(\frac{d_{v,i,j}/\max_{i,j} d_{v,i,j}}{d_{s,i,j}/\max_{i,j} d_{s,i,j} + 10^{-2}}, 1\right) \\
\text{Orthogonality} &= 1 - \frac{2}{N(N-1)}\sum_{i,j,i<j}^{N} \frac{|\langle \phi_i, \phi_j \rangle|}{\|\phi_i\|_2 \|\phi_j\|_2}
\end{aligned}
\tag{9}
$$

where N is the total number of samples, $U(1, N)$ means uniformly sample from $[1, N]$, $d_{s,i,j} = \|\phi_i - \phi_j\|_2$ and $d_{v,i,j} = |\max_a \text{Critic}(\phi_i, a) - \max_a \text{Critic}(\phi_j, a)|$.

Predict State from Visual Representation Besides the three metrics on visual representations and state-values, we further measure the quality of visual representation ϕ_i by predicting the system state s_i only using ϕ_i. The intuition is that a better visual representation should be able to capture the intrinsic system state more precisely. We utilize a two-layer MLP to regress the system state s_i on its corresponding visual representation ϕ_i. Mean squared error is applied to supervise the network as well as to evaluate the network on the test set.

To properly measure all these metrics, We first collect a dataset in cartpole swingup from DMControl using state-based SAC, which is different from any methods we'll benchmark to avoid bias. We run state-based SAC with five random seeds, and take the replay buffer of each run to form a dataset. The whole dataset has $12500 \times 5 = 62500$ state transitions. We measure Dynamics Awareness and Orthogonality on the full dataset, while Diversity is calculated for one run due to computational

cost. For state prediction, we use the first four runs as the training set and the last run is held for the evaluation.

Finally, we benchmark selected methods with five different random seeds on cartpole swingup, and reports the metrics above every 100 model update step. We demonstrate how the four metrics correlate to the environment step and agent performance as Fig. 16 and Fig. 17.

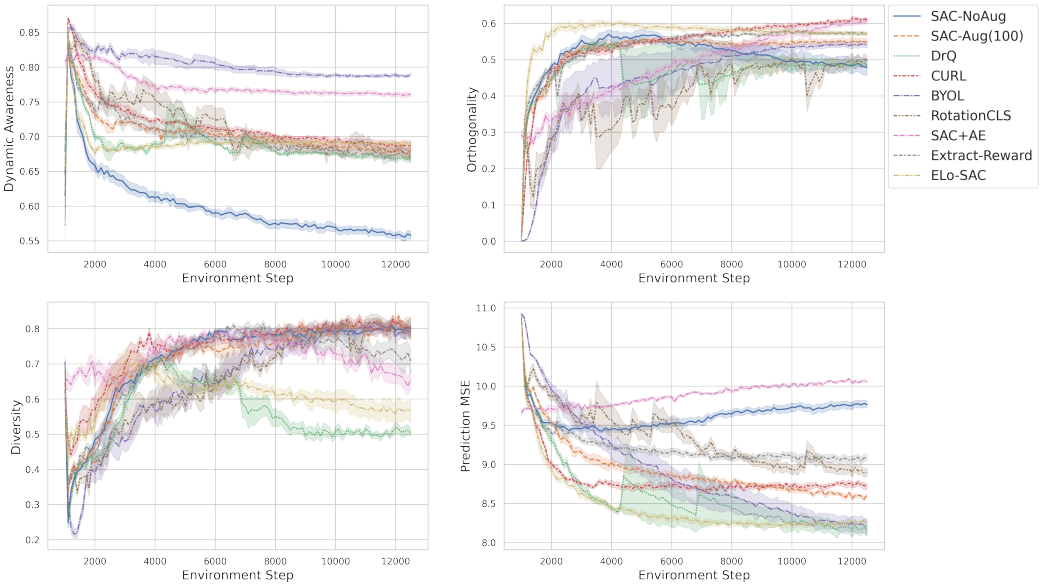


Figure 16: Scores versus representation metric values

Fig. 16 shows how metrics change as training. Most of the methods converge on a similar Orthogonality, Dynamic Awareness, and Diversity value. SAC-NoAug has a low Dynamic Awareness measure which could be used to explain its low performance. While a higher Dynamic Awareness measure does not bring extra scores for BYOL and SAC+AE. Similarly, the lower Diversity value of DrQ and ELo-SAC do not hurt their performance either. Meanwhile, most of the metrics become relatively stable after the first 4000 steps. Therefore, we confirm that the shallower layers of the neural networks in visual reinforcement learning converge faster as observed by Chen et al. [9].

Fig. 17 shows the correlation between metrics and the agent performance. We report the Pearson correlation coefficient as Table 14. As Wang et al. [77] suggested, these metrics only measure certain properties of the visual representation, and they do not suggest that a property is necessary for better policy learning. However, we find that the state prediction error is correlated to the agent performance to some extent, which may be valuable in some cases.

Table 14: Pearson correlation coefficient between scores and representation metrics

Dynamic Awareness	Orthogonality	Diversity	Prediction MSE
-0.284	0.435	0.111	-0.625

A.7 Observation on Pretraining Framework

Besides the joint learning framework used in CURL and SAC+AE, Shelhamer et al. [70] investigate a pretraining framework to combine SSL with RL and use the self-supervised loss as an intrinsic reward to further boost performance during online learning. Recent works on policy learning (e.g., [59, 68, 76, 82, 88]) also take advantage of the self-supervised learning in a multi-step framework and show its great potential in solving challenging visual-based problems.

This pretraining framework is similar to how self-supervision has been benefiting supervised Computer Vision tasks ([8, 10, 12, 17, 34, 40, 60]): pretrain with self-supervised losses, and then finetune

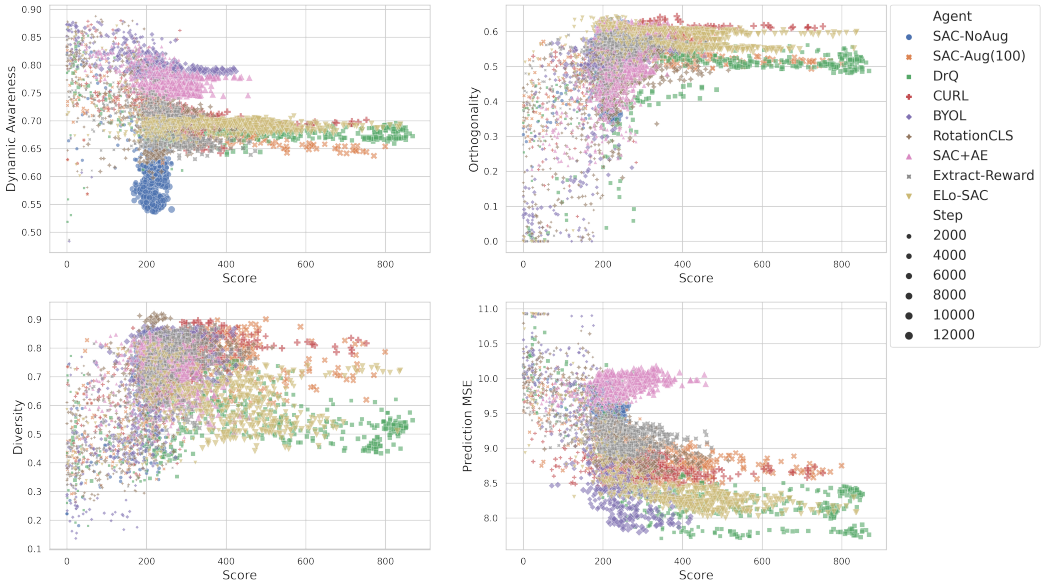


Figure 17: Scores versus representation metric values

with the downstream task loss. Motivated by them, in this section, we design and benchmark the two-stage pretraining framework, replacing the joint learning framework used in CURL and SAC+AE.

In the first stage, we use data collected by training a SAC-Aug(100) agent on the same task and update the visual encoder only using self-supervised loss. We name this stage pretraining which means to use self-supervised losses to update the model and to be downstream task agnostic. Then in the second stage i.e., the online training stage, we only keep the trained encoder from the first stage and train an agent using SAC-Aug(100). The only difference between this stage and training an agent from scratch is that here the visual encoder has been "initialized" with the pre-trained weights while it is randomly initialized in SAC-Aug(100). This also means that the image encoder can be tuned by RL loss in the online training stage to match the online sample distribution. Fig. 18 compares two training frameworks, in which the rounded rectangle means to update the model with the labeled loss for one step.

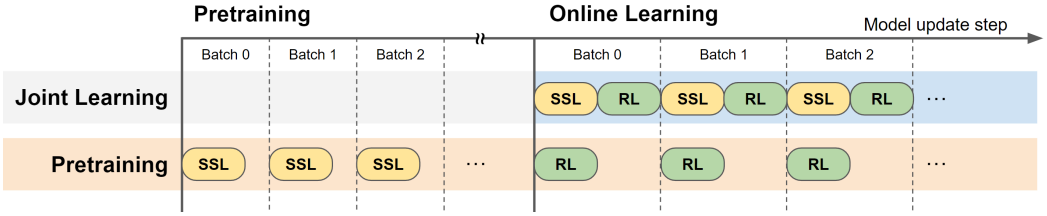


Figure 18: Two learning frameworks for SSL + RL, the rounded rectangle means to update the model with the labeled loss for one step.

The methods using the pretraining framework have the prefix 'Pretrain'. 'Pretrain-Random' means the data used for pretraining is collected by a random policy. In both cases, the pretraining framework has the same model update steps as the joint learning framework baseline. But note that the pretraining model has access to extra data collected by other policies, which makes it an unfair comparison. To this end, we test another joint learning configuration named with the prefix 'Longer'. Here we match the total number of environment steps (or collected data) to its pretraining variants. Similarly, three methods without any self-supervised learning are benchmarked with 'Longer' configuration. We compare two frameworks in six DMControl environments, Relative Scores are reported as Fig. 19 and the full results are shown as Table 15.

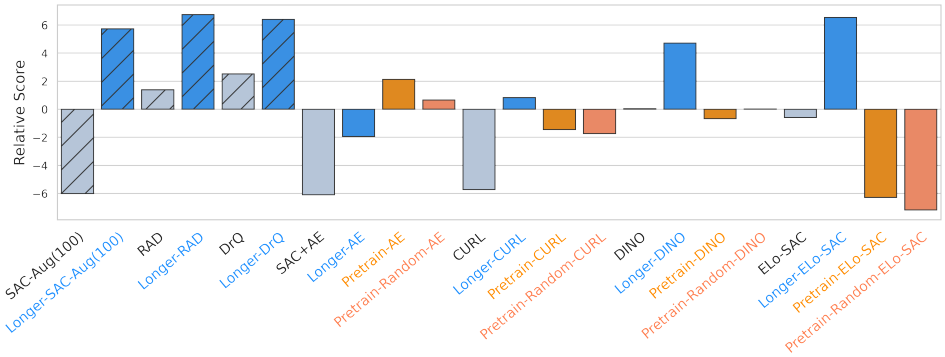


Figure 19: Relative Score of two learning frameworks for combining SSL to RL. The bars with diagonal lines stand for the methods that only use image augmentation without any self-supervised losses.

In general, given the same amount of model updates, the pretraining framework performs better than the joint learning framework except ELo-SAC (we believe this is because ELo-SAC search was done only under the joint learning framework). But such an advantage of the pretraining framework may come from the extra data used in the pretraining stage. When the same amount of data is given, the longer joint-learning configuration usually performs better than the pretraining methods except when AutoEncoder is the self-supervised loss. Such observations imply that the learning framework has different impacts on policy learning even if the same self-supervised loss is applied. It might not be the best practice to directly use the existing self-supervised losses designed for joint learning framework with the pretraining framework. However, only Longer-ELo-SAC achieves comparable results compared to image augmentation based methods with 'Longer' configuration. On the other hand, we argue that on DMControl, the advantages of the pretraining framework come from the access to extra data instead of the framework itself. When the total environment step is limited and no previous data has been collected, the joint learning framework can better solve DMControl problems.

A.8 Limitations

In this paper, we focus on SAC for environments with continuous action space and Rainbow for environments with discrete action space. Though both methods are generic, it will be interesting to see how self-supervised losses work with other RL methods and image augmentations in more challenging environments. Meanwhile, both RAD [46] and DrQ [83] investigate many image augmentation approaches for their learning methods. We only focus on random crop and translate because of their positive effects, and more combinations of image augmentation and self-supervised learning methods worth further investigation. In addition, the search space of ELo-based methods are relatively limited. They may achieve better scores with a larger search space (more losses) and a more representative searching environment.

A.9 Computation Information

Training one DMControl agent for 50k model update steps usually takes 3 hours on one NVIDIA A5000 GPU. It takes around 1.5 hours to train five Atari agents in parallel using an Apple M1 Max CPU.

Table 15: Comparison of the two frameworks. Methods in gray are without a self-supervised loss for reference. The total amount of data/environment step at each stage is listed in the second and the third column.

Agent	Pretraining env.step	Online env.step	ball_in_cup, catch	cartpole, swingup	cheetah, run
SAC-Aug(100)	0	100k	541.4 ± 306.2	563.4 ± 235.0	172.1 ± 64.0
Longer-SAC-Aug(100)	0	200k	944.9 ± 75.3(↑403.5)	851.0 ± 36.4(↑287.6)	424.0 ± 66.8(↑251.9)
RAD	0	100k	879.9 ± 82.0	786.4 ± 95.1	387.9 ± 81.3
Longer-RAD	0	200k	932.6 ± 52.6(↑52.7)	846.2 ± 34.1(↑59.8)	551.4 ± 176.1(↑163.5)
DrQ	0	100k	914.9 ± 21.2	692.2 ± 222.9	360.4 ± 67.7
Longer-DrQ	0	200k	952.0 ± 301.5(↑37.1)	857.4 ± 31.4(↑165.2)	475.9 ± 78.7(↑115.5)
SAC+AE	0	100k	616.1 ± 169.9	388.8 ± 130.1	291.8 ± 59.8
Longer-AE	0	200k	579.4 ± 274.0(↓-36.7)	467.1 ± 196.9(↑78.3)	359.0 ± 57.6(↑67.2)
Pretrain-AE	100k	100k	914.7 ± 129.0(↑298.6)	759.1 ± 99.3(↑370.3)	419.8 ± 41.1(↑128.0)
Pretrain-Random-AE	100k	100k	903.1 ± 219.9(↑287.0)	736.0 ± 97.4(↑347.2)	405.3 ± 55.5(↑113.5)
CURL	0	100k	730.0 ± 179.4	471.5 ± 89.9	215.1 ± 57.3
Longer-CURL	0	200k	935.0 ± 26.5(↑205.0)	776.2 ± 82.2(↑304.7)	307.6 ± 57.3(↑92.5)
Pretrain-CURL	100k	100k	921.0 ± 25.5(↑191.0)	705.4 ± 138.3(↑233.9)	213.0 ± 56.7(↓-2.1)
Pretrain-Random-CURL	100k	100k	874.5 ± 298.3(↑144.5)	745.3 ± 124.5(↑273.8)	224.3 ± 60.6(↑9.2)
DINO	0	100k	916.9 ± 65.7	686.0 ± 152.2	198.3 ± 79.3
Longer-DINO	0	200k	952.6 ± 48.9(↑35.7)	858.1 ± 21.4(↑172.1)	248.6 ± 49.3(↑50.3)
Pretrain-DINO	100k	100k	748.1 ± 164.7(↓-168.8)	759.3 ± 110.8(↑73.3)	344.7 ± 56.5(↑146.4)
Pretrain-Random-DINO	100k	100k	904.6 ± 266.6(↓-12.3)	758.6 ± 86.1(↑72.6)	355.6 ± 77.5(↑157.3)
ELo-SAC	0	100k	888.3 ± 90.6	772.8 ± 167.3	359.7 ± 69.7
Longer-ELo-SAC	0	200k	949.5 ± 29.8(↑61.2)	866.6 ± 30.0(↑93.8)	489.6 ± 149.7(↑129.9)
Pretrain-ELo-SAC	100k	100k	505.8 ± 301.3(↓-382.5)	617.9 ± 147.1(↓-154.9)	400.2 ± 63.6(↑40.5)
Pretrain-Random-ELo-SAC	100k	100k	466.2 ± 200.3(↓-422.1)	519.8 ± 175.4(↓-253.0)	302.9 ± 126.4(↓-56.8)

Agent	Pretraining env.step	Online env.step	finger, spin	reacher, easy	walker, walk
SAC-Aug(100)	0	100k	724.6 ± 154.9	654.4 ± 222.1	422.1 ± 250.8
Longer-SAC-Aug(100)	0	200k	868.6 ± 140.8(↑144.0)	911.6 ± 92.3(↑257.2)	658.3 ± 378.2(↑236.2)
RAD	0	100k	910.4 ± 104.5	508.8 ± 111.5	522.1 ± 95.5
Longer-RAD	0	200k	874.6 ± 150.8(↓-35.8)	819.2 ± 115.7(↑310.4)	765.5 ± 337.8(↑243.4)
DrQ	0	100k	935.6 ± 201.3	713.7 ± 147.6	523.9 ± 182.2
Longer-DrQ	0	200k	906.9 ± 155.7(↓-28.7)	809.2 ± 102.0(↑95.5)	740.0 ± 314.3(↑216.1)
SAC+AE	0	100k	799.0 ± 138.9	481.3 ± 130.4	402.6 ± 161.5
Longer-AE	0	200k	887.8 ± 127.4(↑88.8)	578.6 ± 160.7(↑97.3)	700.3 ± 232.7(↑297.7)
Pretrain-AE	100k	100k	869.8 ± 150.6(↑70.8)	757.8 ± 174.0(↑276.5)	308.0 ± 243.1(↓-94.6)
Pretrain-Random-AE	100k	100k	793.6 ± 175.9(↓-5.4)	858.0 ± 155.0(↑376.7)	107.8 ± 216.1(↓-294.8)
CURL	0	100k	717.8 ± 136.5	569.8 ± 179.4	442.6 ± 87.1
Longer-CURL	0	200k	732.1 ± 146.2(↑14.3)	688.8 ± 229.8(↑119.0)	701.5 ± 148.0(↑258.9)
Pretrain-CURL	100k	100k	785.8 ± 134.1(↑68.0)	754.5 ± 106.2(↑184.7)	277.7 ± 152.0(↓-164.9)
Pretrain-Random-CURL	100k	100k	693.8 ± 178.2(↓-24.0)	804.8 ± 205.8(↑235.0)	356.2 ± 131.8(↓-86.4)
DINO	0	100k	923.1 ± 124.4	686.2 ± 198.2	414.6 ± 162.4
Longer-DINO	0	200k	926.0 ± 128.3(↑2.9)	861.4 ± 131.8(↑175.2)	722.6 ± 251.4(↑308.0)
Pretrain-DINO	100k	100k	877.5 ± 123.8(↓-45.6)	635.0 ± 172.0(↓-51.2)	260.1 ± 145.8(↓-154.5)
Pretrain-Random-DINO	100k	100k	823.4 ± 75.4(↓-99.7)	712.6 ± 126.6(↑26.4)	197.5 ± 147.2(↓-217.1)
ELo-SAC	0	100k	789.3 ± 198.2	478.3 ± 159.9	537.5 ± 164.5
Longer-ELo-SAC	0	200k	919.2 ± 154.1(↑129.9)	753.8 ± 159.9(↑275.5)	789.9 ± 335.3(↑252.4)
Pretrain-ELo-SAC	100k	100k	711.5 ± 161.7(↓-77.8)	503.6 ± 220.1(↑25.3)	115.5 ± 152.4(↓-422.0)
Pretrain-Random-ELo-SAC	100k	100k	742.3 ± 142.4(↓-47.0)	548.0 ± 136.4(↑69.7)	179.7 ± 189.5(↓-357.8)

Figure 20: DMControl score distribution

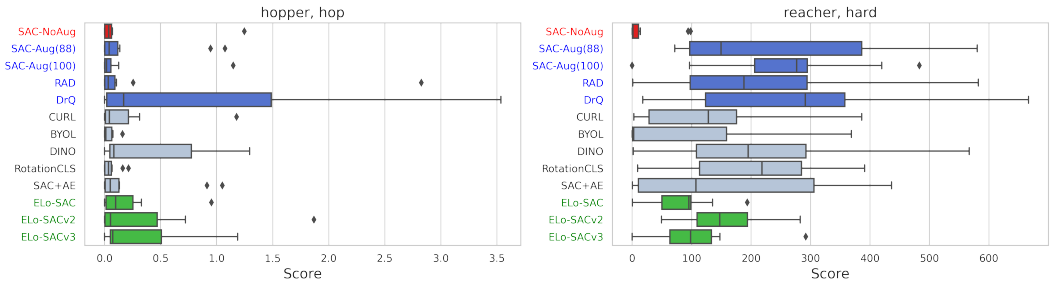


Figure 21: Hard DMControl score distribution

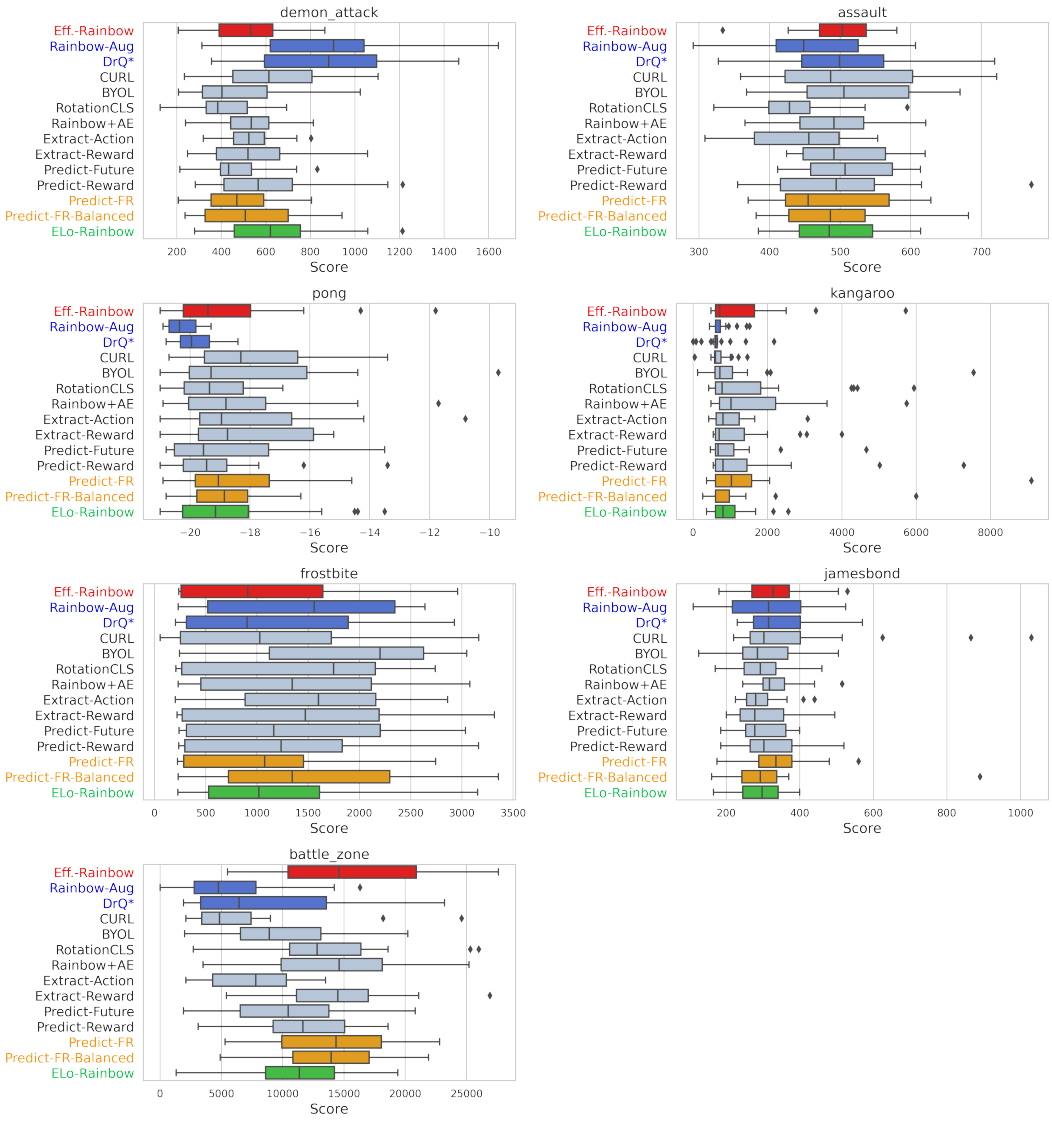


Figure 22: Atari score distribution

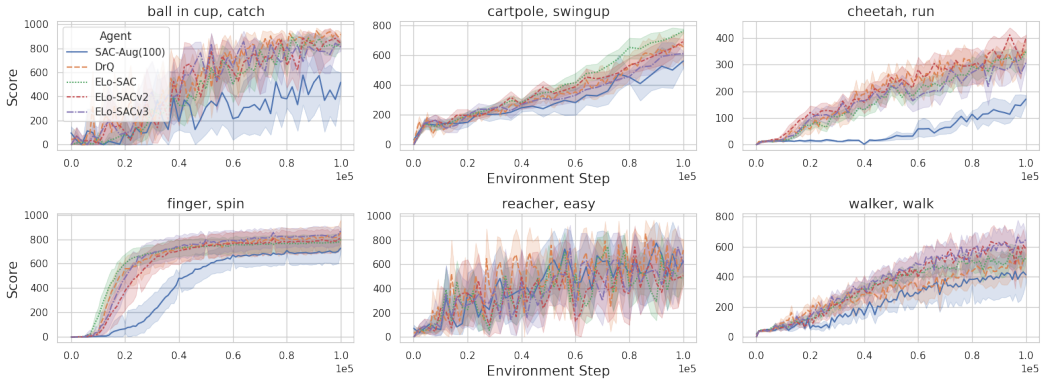


Figure 23: Step-reward curve of ELo-SAC based methods

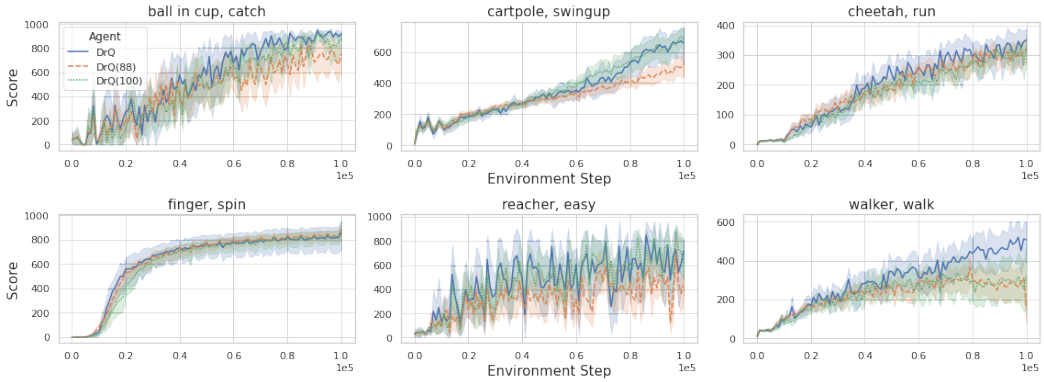


Figure 24: Step-reward curve of DrQ and its variants

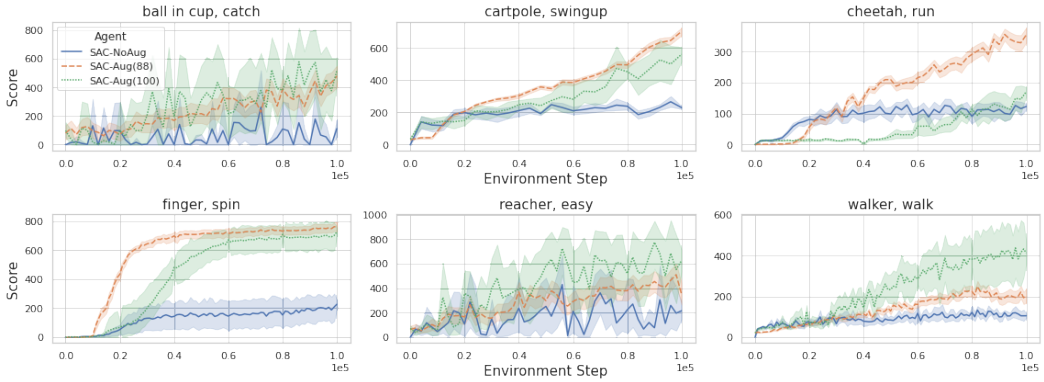


Figure 25: Step-reward curve of SAC variants with different image augmentations

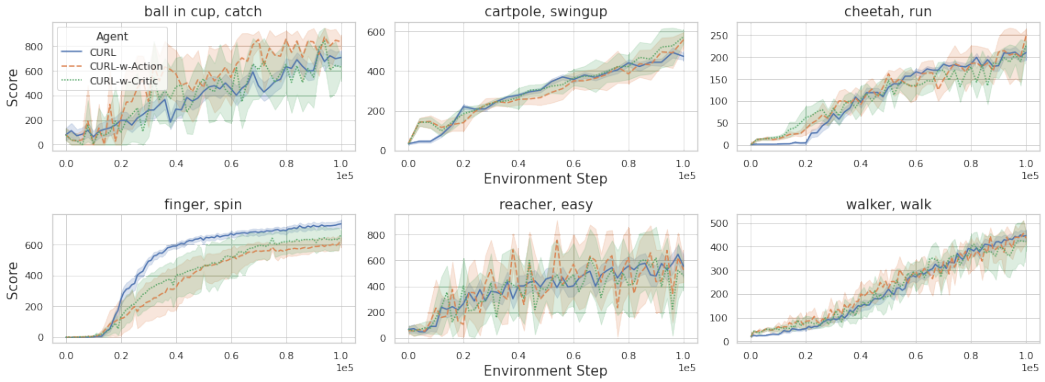


Figure 26: Step-reward curve of CURL and its variants

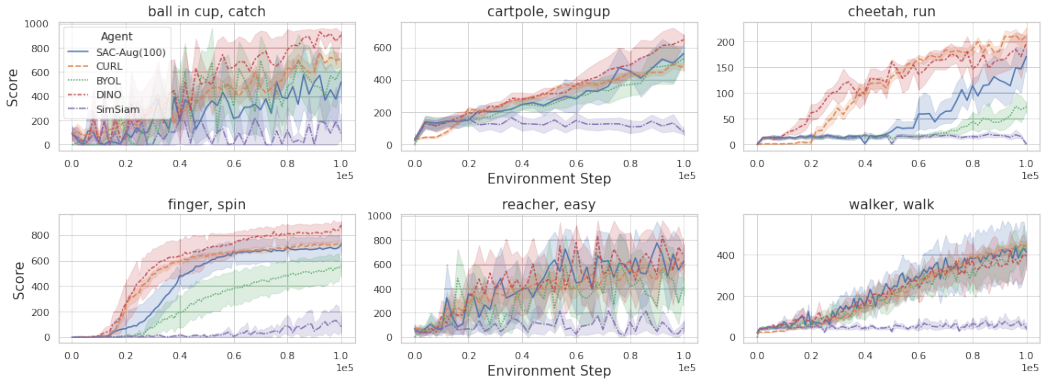


Figure 27: Step-reward curve of self-supervised learning based methods

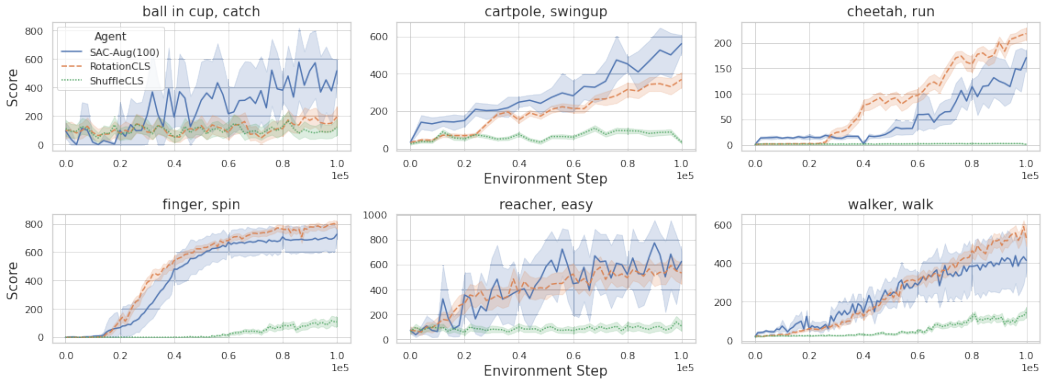


Figure 28: Step-reward curve of classification-based methods (transformation awareness)

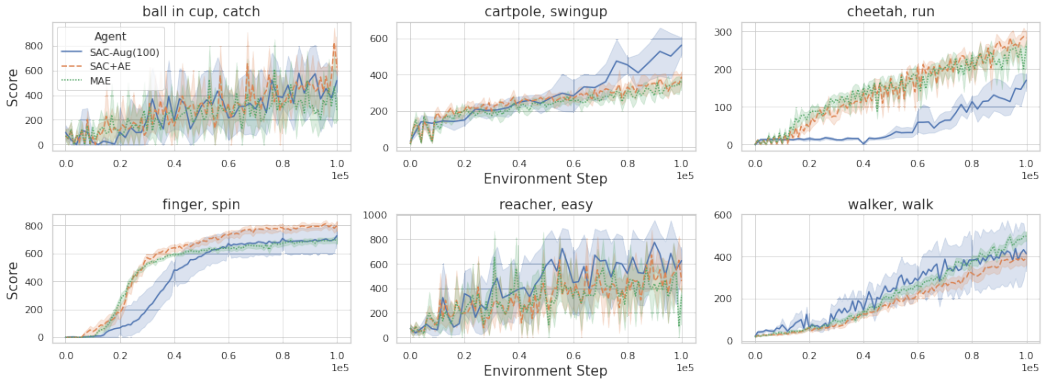


Figure 29: Step-reward curve of reconstruction methods

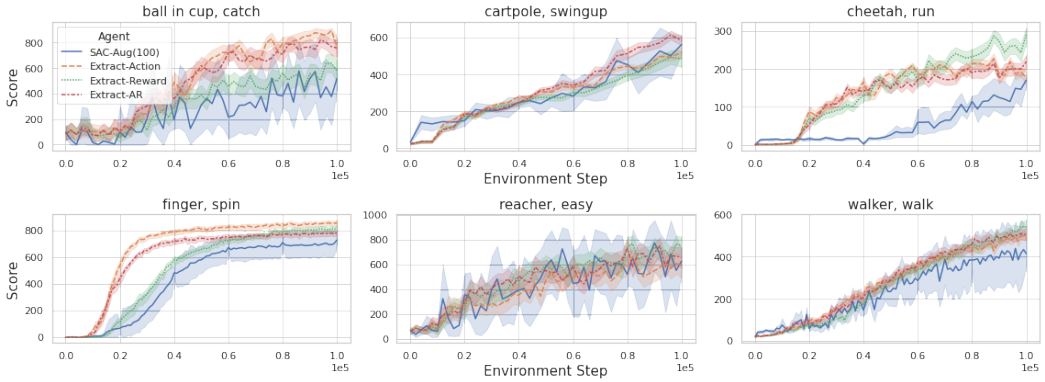


Figure 30: Step-reward curve of RL context prediction methods - 1

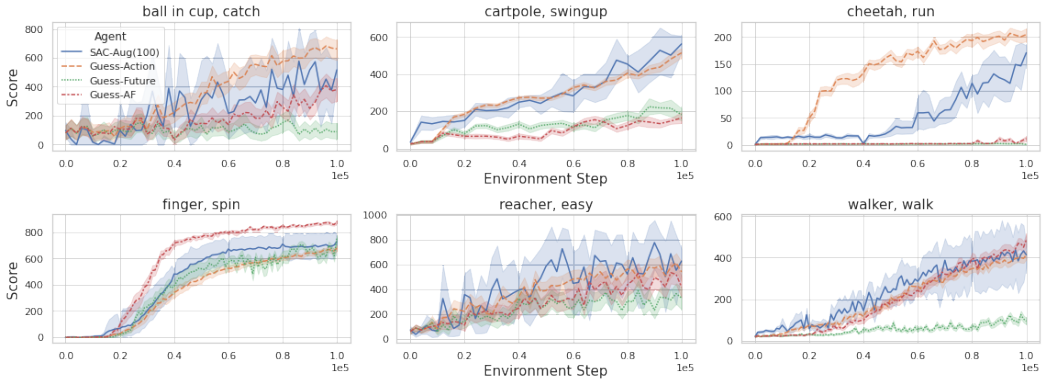


Figure 31: Step-reward curve of RL context prediction methods - 2

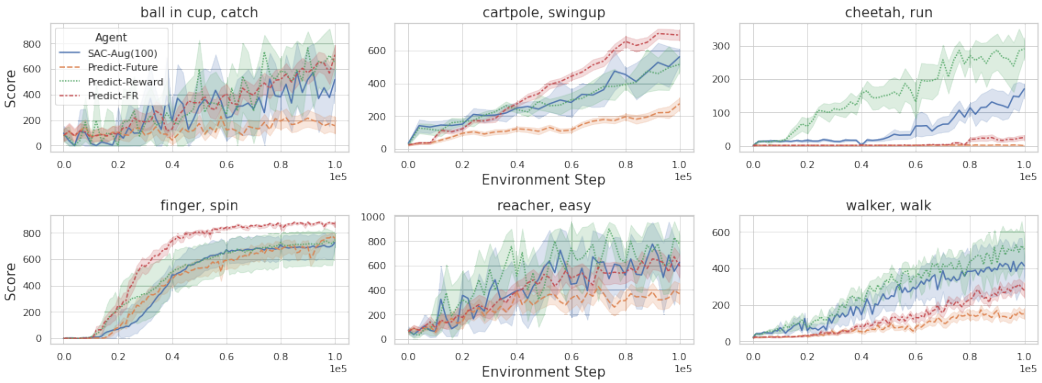


Figure 32: Step-reward curve of RL context prediction methods - 3

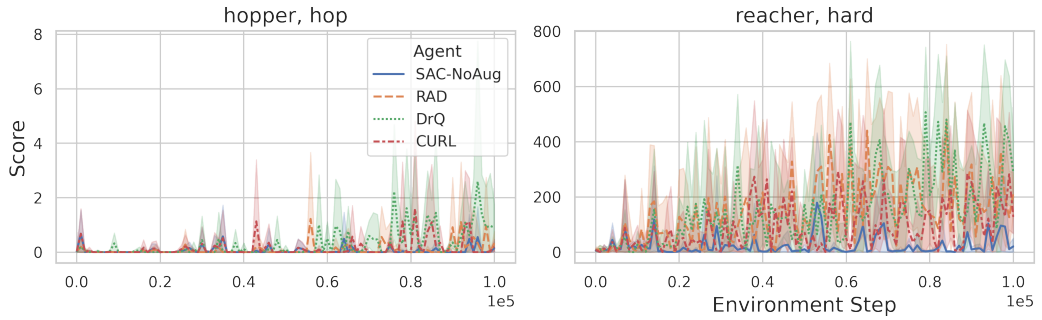


Figure 33: Step-reward curve on two harder DMControl environments - typical methods

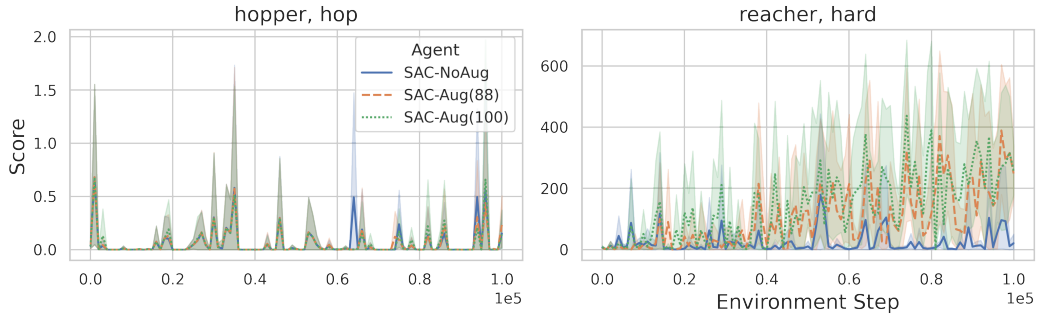


Figure 34: Step-reward curve on two harder DMControl environments - SAC with different image augmentations

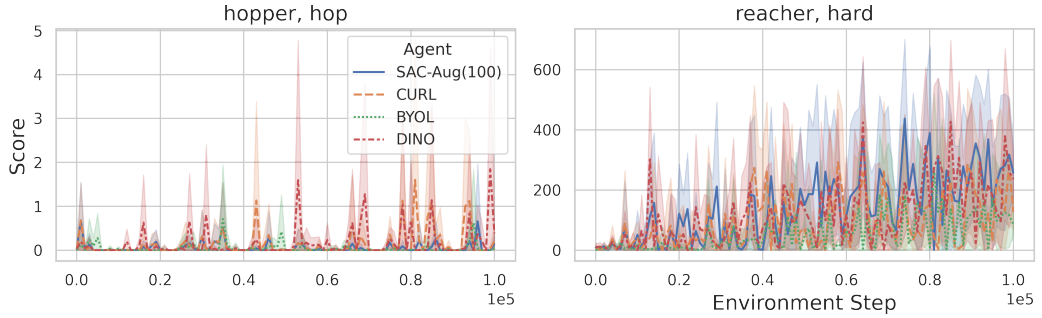


Figure 35: Step-reward curve on two harder DMControl environments - self-supervised learning based methods - 1

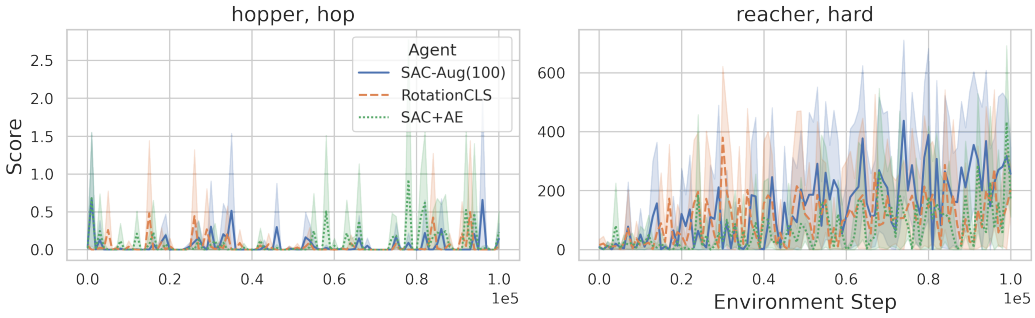


Figure 36: Step-reward curve on two harder DMControl environments - self-supervised learning based methods - 2

Figure 37: Step-reward curve on two harder DMControl environments - ELo-SAC based methods