A. Implementation Details

Below, we explain the implementation details for CURL in the DMControl setting. Specifically, we use the SAC algorithm as the RL objective coupled with CURL and build on top of the publicly released implementation from Yarats et al. (2019). We present in detail the hyperparameters for the architecture and optimization. We do not use any extra hyperparameter for balancing the contrastive loss and the reinforcement learning losses. Both the objectives are weighed equally in the gradient updates.

Table 3. Hyperparameters used for DMControl CURL experiments. Most hyperparameters values are unchanged across environments with the exception for action repeat, learning rate, and batch size.

Hyperparameter	Value
Random crop	True
Observation rendering	$(100, 100)$
Observation downsampling	$(84, 84)$
Replay buffer size	100000
Initial steps	1000
Stacked frames	3
Action repeat	2 finger, spin; walker, walk 8 cartpole, swingup 4 otherwise
Hidden units (MLP)	1024
Evaluation episodes	10
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_\theta, \pi_\psi, Q_\phi)$	$(.9, .999)$
$(\beta_1, \beta_2) \rightarrow (\alpha)$	$(.5, .999)$
Learning rate $(f_\theta, \pi_\psi, Q_\phi)$	$2e-4$ cheetah, run $1e-3$ otherwise
Learning rate (α)	$1e-4$
Batch Size	512 (cheetah), 128 (rest)
Q function EMA τ	0.01
Critic target update freq	2
Convolutional layers	4
Number of filters	32
Non-linearity	ReLU
Encoder EMA τ	0.05
Latent dimension	50
Discount γ	.99
Initial temperature	0.1

Architecture: We use an encoder architecture that is similar to (Yarats et al., 2019), which we sketch in PyTorch-like pseudodocode below. The actor and critic both use the same encoder to embed image observations. A full list of hyperparameters is displayed in Table 3.

For contrastive learning, CURL utilizes momentum for the key encoder (He et al., 2019b) and a bi-linear inner product as the similarity measure (van den Oord et al., 2018). Performance curves ablating these two architectural choices are shown in Figure 5.

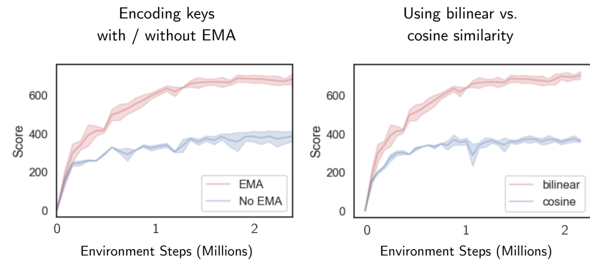

Figure 5. Performance on cheetah-run environment ablated two-ways: (left) using the query encoder or exponentially moving average of the query encoder for encoding keys (right) using the bi-linear inner product as in (van den Oord et al., 2018) or the cosine inner product as in He et al. (2019b); Chen et al. (2020)

Pseudo-code for the architecture is provided below:

```
def encode(x,z_dim):
   """
   ConvNet encoder
   args:
      B-batch_size, C-channels
      H,W-spatial_dims
      x : shape : [B, C, H, W]
      C = 3 * num_frames; 3 - R/G/B
      z_dim: latent dimension
   """

   x = x / 255.

   # c: channels, f: filters
   # k: kernel, s: stride

   z = Conv2d(c=x.shape[1], f=32, k=3, s=2)])(
       x)
   z = ReLU(z)

   for _ in range(num_layers - 1):
      z = Conv2d((c=32, f=32, k=3, s=1))(z)
      z = ReLU(z)

   z = flatten(z)

   # in: input dim, out: output_dim, h:
       hiddens

   z = mlp(in=z.size(),out=z_dim,h=1024)
   z = LayerNorm(z)
   z = tanh(z)
```

Terminology: A common point of confusion is the meaning "training steps." We use the term *environment steps* to denote the amount of times the simulator environment is stepped through and *interaction steps* to denote the number of times the agent steps through its policy. The terms *action repeat* or *frame skip* refer to the number of times an action

is repeated when it's drawn from the agent's policy. For example, if action repeat is set to 4, then 100k interaction steps is equivalent to 400k environment steps.

Batch Updates: After initializing the replay buffer with observations extracted by a random agent, we sample a batch of observations, compute the CURL objectives, and step through the optimizer. Note that since queries and keys are generated by data-augmenting an observation, we can generate arbitrarily many keys to increase the contrastive batch size without sampling any additional observations.

Shared Representations: The objective of performing contrastive learning together with RL is to ensure that the shared encoder learns rich features that facilitate sample efficient control. There is a subtle coincidental connection between MoCo and off-policy RL. Both the frameworks adopt the usage of a momentum averaged (EMA) version of the underlying model. In MoCo, the EMA encoder is used for encoding the keys (targets) while in off-policy RL, the EMA version of the Q-networks are used as targets in the Bellman error (Mnih et al., 2015; Haarnoja et al., 2018). Thanks to this connection, CURL shares the convolutional encoder, momentum coefficient and EMA update between contrastive and reinforcement learning updates for the shared parameters. The MLP part of the critic that operates on top of these convolutional features has a separate momentum coefficient and update decoupled from the image encoder parameters.

Balancing Contrastive and RL Updates: While past work has learned hyperparameters to balance the auxiliary loss coefficient or learning rate relative to the RL objective (Jaderberg et al., 2016; Yarats et al., 2019), CURL does not need any such adjustments. We use both the contrastive and RL objectives together with equal weight and learning rate. This simplifies the training process compared to other methods, such as training a VAE jointly (Hafner et al., 2018; 2019; Lee et al., 2019), that require careful tuning of coefficients for representation learning.

Differences in Data Collection between Computer Vision and RL Settings: There are two key differences between contrastive learning in the computer vision and RL settings because of their different goals. Unsupervised feature learning methods built for downstream vision tasks like image classification assume a setting where there is a large static dataset of unlabeled images. On the other hand, in RL, the dataset changes over time to account for the agent's new experiences. Secondly, the size of the memory bank of labeled images and dataset of unlabeled ones in vision-based settings are 65K and 1M (or 1B) respectively. The goal in vision-based methods is to learn from millions of unlabeled images. On the other hand, the goal in CURL is to develop sample-efficient RL algorithms. For example, to be able to solve a task within 100K timesteps (approximately 2 hours in real-time), an agent can only ingest 100K image frames.

Therefore, unlike MoCo, CURL does not use a memory bank for contrastive learning. Instead, the negatives are constructed on the fly for every minibatch sampled from the agent's replay buffer for an RL update similar to SimCLR. The exact implementation is provided as a PyTorch-like code snippet in 4.7.

Data Augmentation:

Random crop data augmentation has been crucial for the performance of deep learning based computer vision systems in object recognition, detection and segmentation (Krizhevsky et al., 2012; Szegedy et al., 2015; Cubuk et al., 2019; Chen et al., 2020). However, similar augmentation methods have not seen much adoption in the field of RL even though several benchmarks use raw pixels as inputs to the model.

CURL adopts the random crop data augmentation as the stochastic data augmentation applied to a frame stack. To make it easier for the model to correlate spatio-temporal patterns in the input, we apply the same random crop (in terms of box coordinates) across all four frames in the stack as opposed to extracting different random crop positions from each frame in the stack. Further, unlike in computer vision systems where the aspect ratio for random crop is allowed to be as low as 0.08, we preserve much of the spatial information as possible and use a constant aspect ratio of 0.84 between the original and cropped. In our experiments, data augmented samples for CURL are formed by cropping 84×84 frames from an input frame of 100×100.

DMControl: We render observations at 100×100 and randomly crop 84×84 frames. For evaluation, we render observations at 100×100 and center crop to 84×84 pixels. We found that implementing random crop efficiently was extremely important to the success of the algorithm. We provide pseudocode below:

```
from skimage import view_as_windows
import numpy as np

def random_crop(imgs, out):
    """
    Vectorized random crop
    args:
       imgs: shape (B,C,H,W)
       out: output size (e.g. 84)
    """

    # n: batch size.
    n = imgs.shape[0]
    img_size = imgs.shape[-1] # e.g. 100
    crop_max = img_size - out

    imgs = np.transpose(imgs, (0, 2, 3, 1))

    w1 = np.random.randint(0, crop_max, n)
    h1 = np.random.randint(0, crop_max, n)

    # creates all sliding window
    # combinations of size (out)

    windows = view_as_windows(
       imgs, (1, out, out, 1))[..., 0,:,:, 0]
```

```
    # selects a random window
    # for each batch element
    cropped = windows[np.arange(n), w1, h1]
    return cropped
```

B. Atari100k Implementation Details

The flexibility of CURL allows us to apply it to discrete control setting with minimal modifications. Similar to our rationale for picking SAC as the baseline RL algorithm to couple CURL with (for continuous control), we pick the data-efficient version of Rainbow DQN (Efficient Rainbow) (van Hasselt et al., 2019) for Atari100K which performs competitively with an older version of SimPLe (most recent version has improved numbers). In order to understand *specifically* what the gains from CURL are without any other changes, we adopt the *exact* same hyperparameters specified in the paper (van Hasselt et al., 2019) (including a modified convolutional encoder that uses larger kernel size and stride of 5). We present the details in Table 4. Similar to DMControl, the contrastive objective and the RL objective are weighted equally for learning (except for Pong, Freeway, Boxing and PrivateEye for which we used a coefficient of 0.05 for the momentum contastive loss. On a large majority (22 out of 26) of the games, we do not use this adjustment. While it is standard practice to use the same hyperparameters for all games in Atari, papers proposing auxiliary losses have adopted a different practice of using game specific coefficients (Jaderberg et al., 2016).). We use the Efficient Rainbow codebase from `https://github.com/Kaixhin/Rainbow` which has a reproduced version of van Hasselt et al. (2019). We evaluate with 20 random seeds and report the mean score for each game given the high variance nature of the Atari100k steps benchmark. We restrict ourselves to using grayscale renderings of image observations and use random crop of frame stack as data augmentation.

C. Benchmarking Data Efficiency

Tables 1 and 2 show the episode returns of DMControl100k, DMControl500k, and Atari100k across CURL and a number of pixel-based baselines. CURL outperforms all baseline pixel-based methods across experiments on both DMControl100k and DMControl500k. On Atari100k experiments, CURL coupled with Eff Rainbow outperforms the baseline on the majority of games tested (19 out of 26 games).

D. Further Investigation of Data-Efficiency in Contrastive RL

To further benchmark CURL's sample-efficiency, we compare it to state-based SAC on a total of 16 DMControl environments. Shown in Figure 7, CURL matches state-based data-efficiency on most of the environments, but lags behind state-based SAC on more challenging environments.

Table 4. Hyperparameters used for Atari100K CURL experiments. Hyperparameters are unchanged across games.

Hyperparameter	Value
Random crop	True
Image size	$(84, 84)$
Data Augmentation	Random Crop (Train)
Replay buffer size	100000
Training frames	400000
Training steps	100000
Frame skip	4
Stacked frames	4
Action repeat	4
Replay period every	1
Q network: channels	32, 64
Q network: filter size	$5 \times 5, 5 \times 5$
Q network: stride	5, 5
Q network: hidden units	256
Momentum (EMA for CURL) τ	0.001
Non-linearity	ReLU
Reward Clipping	$[-1, 1]$
Multi step return	20
Minimum replay size for sampling	1600
Max frames per episode	108K
Update	Distributional Double Q
Target Network Update Period	every 2000 updates
Support-of-Q-distribution	51 bins
Discount γ	0.99
Batch Size	32
Optimizer	Adam
Optimizer: learning rate	0.0001
Optimizer: $\beta 1$	0.9
Optimizer: $\beta 2$	0.999
Optimizer ϵ	0.000015
Max gradient norm	10
Exploration	Noisy Nets
Noisy nets parameter	0.1
Priority exponent	0.5
Priority correction	$0.4 \rightarrow 1$
Hardware	CPU

E. Ablations

E.1. Learning Temporal Dynamics

To gain insight as to whether CURL learns temporal dynamics across the stacked frames, we also train a variant of CURL where the discriminants are individual frames as opposed to stacked ones. This can be done by sampling stacked frames from the replay buffer but only using the first frame to update the contrastive loss:

```
f_q = x_q[:,:3,...] # (B,C,H,W), C=9.
f_k = x_k[:,:3,...]
```

During the actor-critic update, frames in the batch are en-

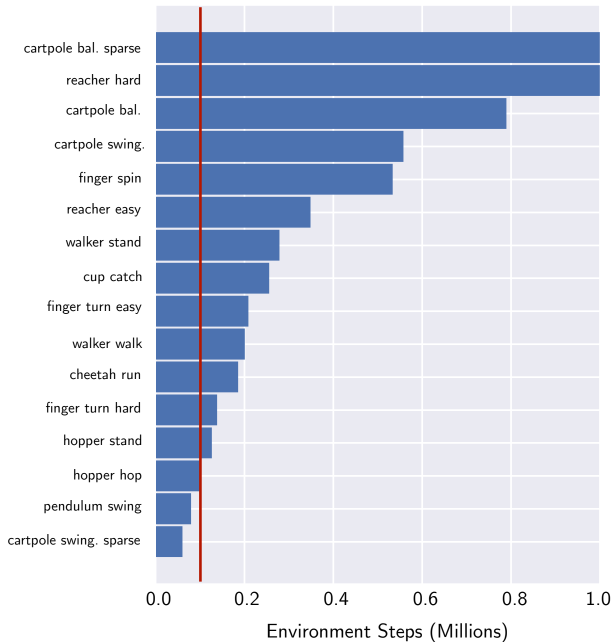

Figure 6. The number of steps it takes a prior leading pixel-based method, Dreamer, to achieve the same score that CURL achieves at 100k training steps (clipped at 1M steps). On average, CURL is 4.5x more data-efficient. We chose Dreamer because the authors (Hafner et al., 2019) report performance for all of the above environments while other baselines like SLAC and SAC+AE only benchmark on 4 and 6 environments, respectively. For further comparison of CURL with these methods, the reader is referred to Table 1 and Figure 4.

coded individually into latent codes, which are then concatenated before being passed to a dense network.

```
# x: (B,C,H,W), C=9.
z1 = encode(x[:,:3,...])
z2 = encode(x[:,3:6,...])
z3 = encode(x[:,6:9,...])
z = torch.cat([z1,z2,z3],-1)
```

Encoding each frame indiviudally ensures that the contrastive objective only has access to visual discriminants. Comparing the visual and spatiotemporal variants of CURL in Figure 8 shows that the variant trained on stacked frames outperforms the visual-only version in most environments. The only exceptions are reacher and ball-in-cup environments. Indeed, in those environments the visual signal is strong enough to solve the task optimally, whereas in other environments, such as walker and cheetah, where balance or coordination is required, visual information alone is insufficient.

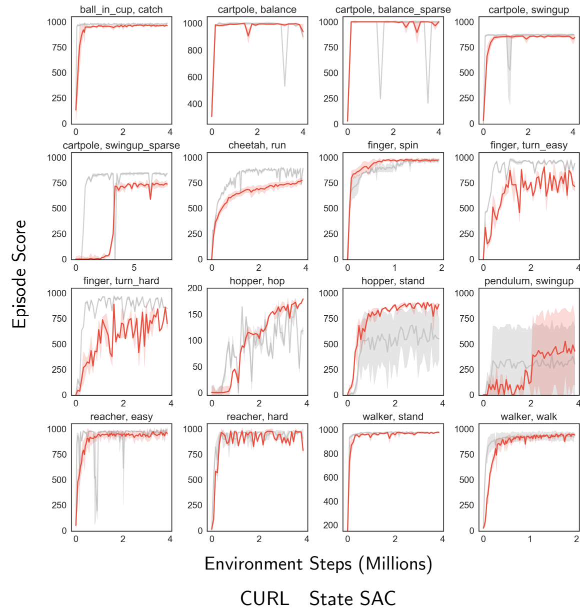

Figure 7. CURL compared to state-based SAC run for 3 seeds on each of 16 selected DMControl environments. For the 6 environments in 4, CURL performance is averaged over 10 seeds.

E.2. Increasing Gradient Updates per Agent Step

Although most baselines we benchmark against use one gradient update per agent step, it was recently empirically shown that increasing the ratio of gradients per step improves data-efficiency in RL (Kielak, 2020). This finding is also supported by SLAC (Lee et al., 2019), where results are shown with a ratio of 1:1 (SLACv1) and 3:1 (SLACv2). We

Table 5. Scores achieved by CURL and SLAC when run with a 3:1 ratio of gradient updates per agent step on DMControl500k and DMControl100k. CURL achieves state-of-the-art performance on the majority (**3** out of **4**) environments on DMControl500k. Performance of both algorithms is improved relative to the 1:1 ratio reported for all baselines in Table 1 but at the cost of significant compute and wall-clock time overhead.

DMControl500k	CURL	SLACv2
Finger, spin	$\mathbf{923 \pm 50}$	884 ± 98
Walker, walk	$\mathbf{911 \pm 35}$	891 ± 60
Cheetah, run	545 ± 39	$\mathbf{791 \pm 37}$
Ball in cup, catch	$\mathbf{948 \pm 21}$	885 ± 154
DMControl100k	**CURL**	**SLACv2**
Finger, spin	$\mathbf{741 \pm 118}$	728 ± 212
Walker, walk	428 ± 59	$\mathbf{513 \pm 41}$
Cheetah, run	314 ± 46	$\mathbf{438 \pm 76}$
Ball in cup, catch	$\mathbf{899 \pm 47}$	837 ± 147

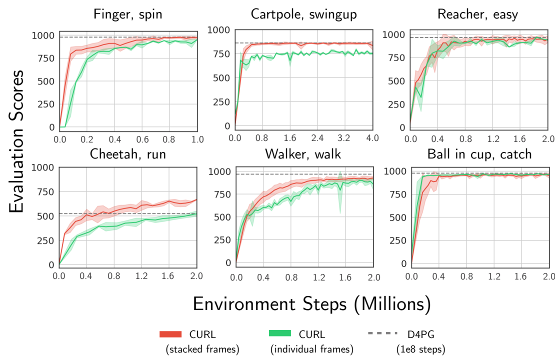

Figure 8. CURL with temporal and visual discrimination (red) compared to CURL with only visual discrimination (green). In most settings, the variant with temporal variant outperforms the purely visual variant of CURL. The two exceptions are reacher and ball in cup environments, suggesting that learning dynamics is not necessary for those two environments. Note that the walker environment was run with action repeat of 4, whereas walker walk in the main results Table 1 and Figure 7 was run with action repeat of 2.

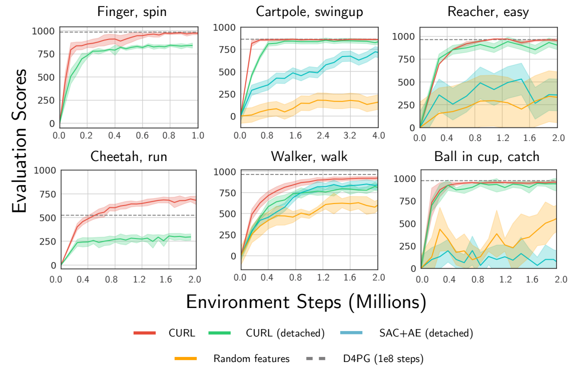

Figure 9. CURL where the CNN part of the encoder receives gradients from both the contrastive loss and critic (red) compared to CURL with the convolutional part of the encoder trained only with the contrastive objective (green). The detached encoder variant is able to learn representations that enable near-optimal learning on most environments, except for cheetah. As in Figure 8, the walker environment was run with action repeat of 4.

E.3. Decoupling Representation Learning from Reinforcement Learning

Typically, Deep RL representations depend almost entirely on the reward function specific to a task. However, hand-crafted representations such as the proprioceptive state are independent of the reward function. It is much more desirable to learn reward-agnostic representations, so that the same representation can be re-used across different RL tasks. We test whether CURL can learn such representations by comparing CURL to a variant where the critic gradients are backpropagated through the critic and contrastive dense feedforward networks but stopped before reaching the convolutional neural network (CNN) part of the encoder.

Scores displayed in Figure 9 show that for many environments, the detached CNN representations are sufficient to learn an optimal policy. The major exception is the cheetah environment, where the detached representation significantly under-performs. Though promising, we leave further exploration of task-agnostic representations for future work.

E.4. Predicting State from Pixels

Despite improved sample-efficiency on most DMControl tasks, there is still a visible gap between the performance of SAC on state and SAC with CURL in some environments. Since CURL learns representations by performing instance discrimination across stacks of three frames, it's possible that the reason for degraded sample-efficiency on more challenging tasks is due to partial-observability of the ground truth state.

To test this hypothesis, we perform supervised regression (X, Y) from pixels X to the proprioceptive state Y, where each data point $x \in X$ is a stack of three consecutive frames and $y \in Y$ is the corresponding state extracted from the simulator. We find that the error in predicting the state from pixels correlates with the policy performance of pixel-based methods. Test-time error rates displayed in Figure 10 show that environments that CURL solves as efficiently as state-based SAC have low error-rates in predicting the state from stacks of pixels. The prediction error increases for more challenging environments, such as cheetah-run and walker-walk. Finally, the error is highest for environments where current pixel-based methods, CURL included, make no progress at all (Tassa et al., 2018), such as humanoid and swimmer.

This investigation suggests that degraded policy performance on challenging tasks may result from the lack of requisite information about the underlying state in the pixel data used for learning representations. We leave further investigation for future work.

E.5. CURL + Efficient Rainbow Atari runs

We report the scores (Tables 6 and 7) for 20 seeds across the 26 Atari games in the Atari100k benchmark for CURL cou-

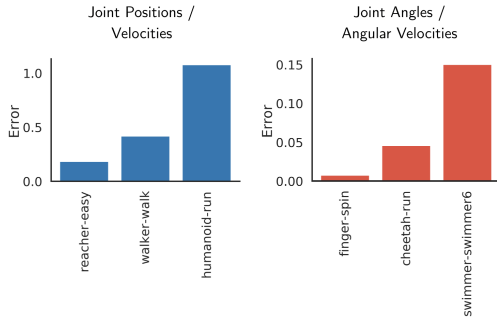

Figure 10. Test-time mean squared error for predicting the proprioceptive state from pixels on a number of DMControl environments. In DMControl, environments fall into two groups - where the state corresponds to either (a) positions and velocities of the robot joints or (b) the joint angles and angular velocities.

pled with Efficient Rainbow. The variance across multiple seeds is considerably high in this benchmark. Therefore, we report the scores for each of the seeds along with the mean and standard deviation for each game.

Pacman	Frostbite	Asterix	KungFuMaster	Kangaroo	Gopher	RoadRunner	JamesBond	BattleZone	Seaquest	Assault	Krull	Qbert
1287	2292	850	8470	600	1036	2820	305	18100	322	634.2	3404.3	1020
1608	1046	525	10870	2280	574	3190	265	18200	236	696.8	2443.5	650
1466	1209	655	10920	1940	540	7840	335	26800	352	655.2	6791.4	830
1430	255	565	7730	1140	618	12060	145	21300	386	443	3022.5	902.5
1114	426	715	17525	520	534	8340	565	7900	458	546	3892.2	3957.5
1083	2280	715	3560	600	596	6920	565	8100	224	564.9	3505.5	772.5
2301	259	770	10940	600	502	2230	350	12000	282	514.4	2564.1	782.5
1128	335	980	23420	900	998	4250	365	16500	339	516.6	4079.7	727.5
1184	1409	665	15160	600	950	1570	140	23900	526	661.5	2376.4	705
1510	258	610	15370	730	544	6300	425	19900	436	664.5	4161.8	757.5
2343	335	905	22260	600	796	3100	315	10000	272	529	3311.1	647.5
1063	1062	800	17320	880	522	1060	335	11200	428	445.2	2517.3	562.5
2040	1542	675	31820	220	392	6050	735	9700	358	573.3	3764.7	2425
1195	1102	795	23360	920	780	11810	950	23500	533	531.3	10150.2	1112.5
1343	2461	585	27460	600	792	4630	520	10500	968	663.6	2883.6	527.5
1354	257	865	7770	2300	454	2530	755	18100	314	795.3	5123.7	472.5
1925	513	730	8820	320	564	6840	750	9000	378	633	3652.5	610
1228	1826	680	2980	600	522	6580	795	8900	168	674.1	2376.4	697.5
1099	1889	965	10100	600	496	10720	450	10700	242	604.8	11745	1847.5
1608	2869	640	10300	500	1176	4380	355	13100	467	665.7	2826	840
1465.5	1181.3	734.5	14307.8	872.5	669.3	5661	471	14870	384.5	600.6	4229.6	1042.4
397.5	856.2	129.8	7919.3	600.1	220.6	3289.3	226.2	5964.3	170.2	89.5	2540.6	828.4

Table 6. CURL implemented on top of Efficient Rainbow - Scores reported for 20 random seeds for each of the above games, with the last two rows being the mean and standard deviation across the runs.

UpNDown	Hero	CrazyClimber	ChopperComm.	DemonAttack	Amidar	Alien	BankHeist	Breakout	Freeway	Pong	PrivateEye	Boxing
3529	8747.5	19090	560	611.5	150.9	616	95	3.6	29.2	-19.3	100	-0.5
772	3026	8290	1530	707.5	131.2	923	184	5	25.4	-16.9	100	-11.4
5972	7146	12160	1390	843.5	141.5	467	75	3.2	27.6	-12	100	4
2793	7686	8920	1100	330.5	133.7	441	232	5.1	28.6	-19.6	100	3.6
3546	7335	11360	500	759	157.1	716	187	2.9	22.8	-17.8	1357.4	6.2
4552	7325	4110	990	940	125.4	453	367	6.3	29.6	-18.9	100	5
2972	7275.5	9460	780	1136	183.2	273	186	5.9	23.3	-15.9	0	-1.7
2865	3115	20630	1180	758	153.6	540	68	2.6	27.6	-15.2	100	0.1
3098	7424	6780	1380	772.5	127.8	499	60	5.9	26.1	-18.7	100	3.5
1953	7475	13570	970	820	149.4	475	123	4.3	28.3	-13.3	100	-0.5
1467	3135	11890	1200	784	125.7	553	72	3.2	21.8	-17.2	1510	-22.1
2912	5060.5	9160	1130	1080	130.4	446	53	4.8	21.8	-20.1	100	-1.8
4123	4409	10960	1380	847	133	533	68	6.3	28.9	-16.5	100	1.6
2334	6979	17360	1230	771.5	140.5	968	36	7.3	28.2	-14.9	100	3.6
2605	4159	8930	1350	907.5	133.8	499	53	4.8	28.3	-19.3	100	-17.6
2432	7560	11510	1080	1095.5	191.8	523	105	3.7	26.8	-15.6	0	21.7
3826	8587	22690	1210	700	115.5	616	276	6.6	27.5	-21	100	2
3052	4683.5	8120	840	803.5	164	475	69	5.5	26.5	-10.5	0	5.9
3131	7317	13500	730	818	131.7	525	50	4.3	26.8	-13.3	100	18.7
1169	7141	14440	640	866	122.4	622	273	6.2	28.6	-13.1	100	3.7
2955.2	6279.3	12146.5	1058.5	817.6	142.1	558.2	131.6	4.9	26.7	-16.5	218.4	1.2
1181.1	1871.5	4765.6	299.1	176.6	20.0	160.3	94.4	1.4	2.4	2.9	417.9	10.0

Table 7. CURL implemented on top of Efficient Rainbow - Scores reported for 20 random seeds for each of the above games, with the last two rows being the mean and standard deviation across the runs.