Local-guided Global: Paired Similarity Representation for Visual Reinforcement Learning - Supplementary material

Hyesong Choi[1], Hunsang Lee[2], Wonil Song[3], Sangryul Jeon[4], Kwanghoon Sohn[3], Dongbo Min[1†]
[1]Ewha W. University [2]Hyundai Motor Company [3]Yonsei University [4]University of Michigan

In this document, we provide more comprehensive results and detailed information not provided in the original manuscript due to the page limit as below.

We include the following material:

1. Implementation Details

This section provides tables summarizing the hyperparameters used for experiments on Atari Games [5] and DMControl Suite [12] and the detailed description of our network architecture.

1.1. Hyperparameters on Atari Games and DMControl Suite

For the reproducibility of our work, we provide full hyperparameters for our experiments on Atari Games [5] in Table 1. We follow the common practices used to set up Rainbow DQN [14] in existing methods [6, 10] for experimenting on Atari Games. Table 2 shows a full list of hyperparameters for DMControl suites experiments. We utilize the similar hyperparameters and optimizer to CURL [6].

1.2. Network Architecture

The detailed network architecture of our method is provided in Table 3. We describe the architectures of the encoder, correspondence estimation module and global prediction module. 'N', 'K' and 'S' of convolution operations represent the channel, kernel size, and stride, respectively. 'LReLU' and 'BN' indicate LeakyReLU [8] and Batch Normalization [4]. We also provide input and output tensors of each layer for better understanding.

Table 1. Hyperparameters used for PSRL experiments on Atari Games

Parameter	Value
Observation Size	(84, 84)
Augmentation	Random shifts ($\pm$4 pixels), Intensity (scale=0.05)
Image Gray-scale	True
Update	Distributional Q
Stacked Frames	4
Action Repeat	4
Reward Clipping	[-1, 1]
Training Steps	100K
Evaluation Trajectories	100
Minimum Replay Size (for sampling)	2000
Max Frames (per episode)	108K
Support Of Q-Distribution	51 bins
Discount Factor	0.99
Optimizer	Adam
Optimizer: learning rate	0.0001
Optimizer: β_1	0.9
Optimizer: β_2	0.999
Optimizer: ϵ	0.00015
Max Gradient Norm	10
Multi Step Return	10
Target Network: Update Period	1
Q Network: Channels	32, 64, 64
Q Network: Filter Size	8×8, 4×4, 3×3
Q Network: Stride	(4, 4), (2, 2), (1, 1)
Q Network: Hidden Units	256
Non-Linearity	ReLU
Replay Period Every	1
Updates Per Step	2
Exploration	Noisy Nets
Noisy Nets Parameter	0.5
Priority Exponent	0.5
Priority Correction	$0.4 \to 1$

Table 2. Hyperparameters used for PSRL experiments on DMControl

Parameter	Value
Observation Size	(84, 84)
Observation Rendering	(100, 100)
Augmentation	Random crop, translation
Stacked Frames	3
Action Repeat	2 (finger-spin, walker-walk), 8 (cartpole-swingup), 4 (otherwise)
Evaluation Episodes	10
Discount Factor	0.99
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_\theta, \pi_\psi, Q_\phi)$	(0.9, 0.999)
$(\beta_1, \beta_2) \rightarrow (\alpha)$	(0.5, 0.999)
Learning Rate $(f_\theta, \pi_\psi, Q_\phi)$	$2e-4$ (cheetah-run), $1e-3$ (otherwise)
Learning Rate (α)	$1e-4$
Batch Size	64
Replay Buffer Size	100000
Initial Steps	1000
Hidden Units (MLP)	1024
Q Function EMA τ	0.01
Critic Target Update Frequency	2
Convolutional Layers	4
Number Of Filters	32
Non-Linearity	ReLU
Latent Dimension	50
Initial Temperature	0.1

1.3. Environments

PSRL was implemented in Pytorch [9] with the use of rlpyt [11] and Mujoco [13] license, and was simulated on 1 Titan RTX GPU.

1.4. Details of using correspondence maps

Use of internal correspondence: When four consecutive frames are used in Rainbow DQN ($M = 3$), the external correspondences are specified using query images o_k ($I_k \sim I_{k+3}$) and target images o_{k+1} ($I_{k+1} \sim I_{k+4}$) as follows:

- correspondence (z^q_k, z^t_{k+1})
- correspondence (z^q_{k+1}, z^t_{k+2})
- correspondence (z^q_{k+2}, z^t_{k+3})
- correspondence (z^q_{k+3}, z^t_{k+4}).

The internal correspondence is computed using query images o_k as

- correspondence (z^q_k, z^q_{k+3}).

The external correspondences computed between consecutive two frames are used to transform the query representations. The internal correspondence is also predicted using the same self-supervised correspondence estimation module yet with two distant frames ($z^q_k \rightarrow z^q_{k+3}$). We found that this is often effective in dealing with the case where the external spatial difference between two consecutive frames is relatively small. Namely, the internal correspondence between k^{th} and $(k+3)^{th}$ frames can be complementary when the external correspondences are rather small and thus correspondence learning becomes less effective.

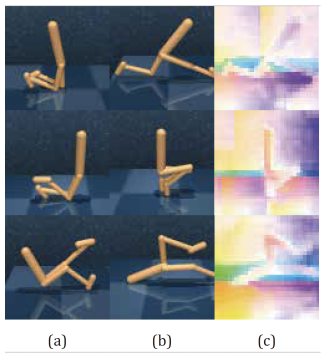

Figure 1. Visualization of the dense correspondence maps learned by PSRL: (a) The source frame, (b) the target frame, and (c) the dence correspondence map of the task 'Walker, Walk' of the DM Control Suite.

The internal correspondence is computed with a stack of images o_k, while the external correspondences are computed between o_k and o_{k+1}. This is why we name two correspondence parts 'internal' correspondence and 'external' correspondence, respectively. Note that the internal correspondence is computed only between two distant frames (k^{th} and $(k+3)^{th}$ frames in o_k), considering that the external correspondence is computed between two consecutive frames as above.

Image and representation transform: The representation we use contains spatial information. For instance, suppose the convolutional feature map z of the size $32 \times 32 \times 64$ is generated from an input image I of the size $128 \times 128 \times 3$. The feature map can have a lower spatial resolution (by the downsampling operator such as max-pooling or strided convolution) and a larger channel dimension than the input image. The 64 dimensional vector at a spatial position of the feature map represents the information of corresponding positions of the input image. Therefore, we can employ the correspondence in image transform for warping the feature map (query representations). This kind of implementation has been commonly used in many computer vision tasks where an image alignment is required.

Visualization of dense correspondence maps: For a better understanding of the proposed method, the examples of the dense correspondence maps predicted by PSRL are represented in Figure 1.

Table 3. Detailed description of the proposed network architecture

Encoder			
Layer	Operations	Input	Output
1	Conv(N32, K8, S4) - ReLU - DropOut	$\mathbf{I_t}\sim\mathbf{I_{t+M}}$	$en1_t\sim en1_{t+M}$
2	Conv(N64, K4, S2) - ReLU - DropOut	$en1_t\sim en1_{t+M}$	$en2_t\sim en2_{t+M}$
3	Conv(N64, K3, S1) - ReLU - DropOut	$en2_t\sim en2_{t+M}$	$en3_t\sim en3_{t+M}$
Correspondence Estimation Module			
Layer	Operations	Input	Output
4	Conv(N32, K1, S1) - BN - LReLU	$en3_t$	$\tilde{en3}_t$
5	Compute Correlation Volume	$en3_t$, $en3_{t+k}$	corr
6	Concatenation	$\tilde{en3}_t$, corr	conc0
5	Conv(N256, K3, S1) - BN - LReLU	conc0	conv1
6	Conv(N512, K3, S1) - BN - LReLU	conv1	conv2p
7	Conv(N512, K3, S1) - BN - LReLU	conv2p	conv2
8	Conv(N512, K3, S1) - BN - LReLU	conv2	conv3p
9	Conv(N512, K3, S1) - BN - LReLU	conv3p	conv3
10	Conv(N2, K3, S1)	conv3	correspondence3
11	Upsampling	correspondence3, conv2	correspondence3up, conv2up
12	Deconv(N256, K4, S2) - LReLU	conv3	conv3d
13	Concatenation	conv2up, conv3d, correspondence3up	conc3
14	Conv(N2, K3, S1)	conc3	correspondence2
15	Upsampling	correspondence2, conv1	correspondence2up, conv1up
16	Deconv(N128, K4, S2) - LReLU	conc3	conc3d
17	Concatenation	conv1up, conc3d, correspondence2up	conc2
18	Conv(N2, K3, S1)	conc2	correspondence1
19	Upsampling	correspondence1, $en2_t$	correspondence1up, $en2up_t$
20	Deconv(N130, K4, S2) - LReLU	conc2	conc2d
21	Concatenation	$en2up_t$, conc2d, correspondence1up	conc1
22	Conv(N2, K3, S1)	conc1	correspondence0
Global Prediction Model			
Layer	Operations	Input	Output
1	Onehot Encoding (Only for Atari)	a_k	a_k
2	Concatenate	Z^q_k, a_k	conc
3	Conv(N256, K3, S1) - ReLU	conc	conv1t
4	Conv(N64, K3, S1) - ReLU	conv1t	conv2t
5	Min-Max Normalize	conv2t	Z^p_{k+1}

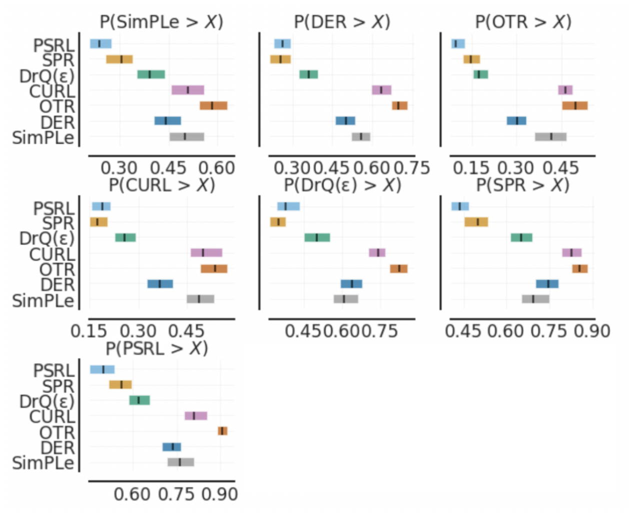

Figure 2. Evaluation result of 'Probability of Improvement' proposed in Agarwal *et al.* [1].

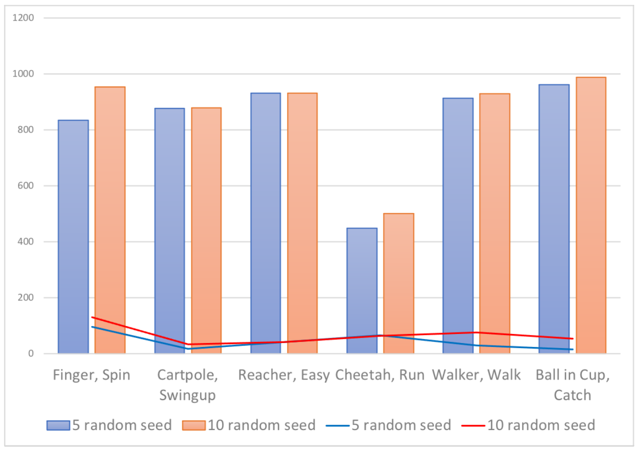

Figure 3. Quantitative evaluation on the DMControl Suite [12] according to the number of the random seed: We show the mean (bar) and standard deviation (line) of episode return on the DMControl Suite [12] after 500K time steps for each 5 random seeds and 10 random seeds to show the consistency of the proposed method regardless of random seeds.

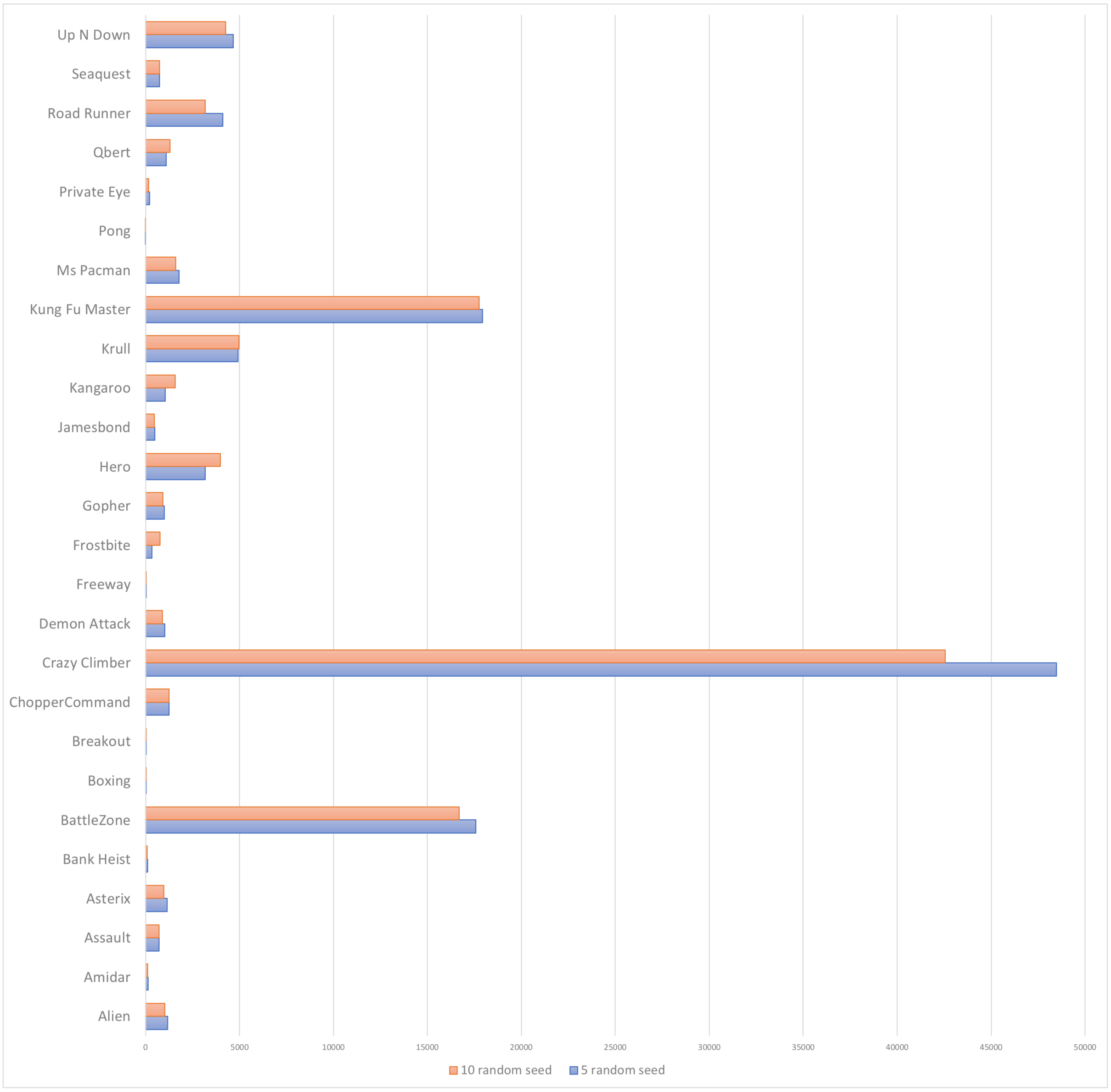

Figure 4. Quantitative evaluation on the 26 Atari games [5] according to the number of the random seeds: We show the mean performance on the 26 Atari games [5] after 100K time steps for each 5 random seeds and 10 random seeds to show the consistency of the proposed method regardless of random seeds.

2. Additional evaluation metric

In all experiments, the evaluation on Atari Games [5] was conducted by measuring the performance with 10 or more random seeds, following the previous studies including SPR [10] and CURL [6]. Recently, Agarwal *et al.* [1] analyzes the problems related to statistical uncertainty in the existing evaluation method of Atari Games [5]. Accordingly, we added the results of a more complete evaluation using 'Probability of Improvement' proposed in Agarwal *et al.* [1] in Figure 2.

This evaluation method estimates how likely an algorithm improves upon another algorithm [1]. For instance, 'P(SimPLe>X)' indicates the probability (written in the horizontal line) that 'SimPLe' is better than another algorithm, called 'X', which is listed in the vertical line (e.g., PSRL, SPR,..., DER). Namely, the probability that SimPLe is better than SPR, 'P(SimPLe>SPR)', has an average value of about 0.3.

In this context, in the six graphs above, the smaller the value of PSRL, the higher the performance. Also, in the graph of PSRL below, 'P(PSRL > X)', PSRL ranks the highest as the remaining bars of algorithms are located to the right of the bar of PSRL. From this analysis, it can be reconfirmed that PSRL has the most superior performance compared to the state-of-the-art methods on Atari Games [5] as mentioned in the result of Section 4.1.

3. Performance consistency evaluation

We consider that the performance depends on the choice of the seed, so we measure the performance by using 10 random seeds and averaging results. To prove that our performance improvement is consistent and not caused by the noise of the estimation, we observed the change in the average performance according to the number of the random seeds. Figure 3 and 4 represent the quantitative evaluation on the DMControl Suite [12] and Atari games [5] according to the number of the random seed. We show result of 5 random seeds and 10 random seeds to compare the performance. In Figure 3, we additionally showed the standard deviation of the performance. The bar graph represents the average of the episode return, and the line graph represents the standard deviation.

From Figure 3 and 4, it can be seen that our results do not differ significantly depending on the number of random seeds. It can be seen that our performance is significantly higher than the SOTA performance regardless of random seeds, and is consistent.

4. Detailed explanations

4.1. Paired Similarity Representation

	Correspondence Aware Transform (CAT)	**Action Aware Transform (AAT)**
Local head	$\mathcal{L}_1$ (3D volume)	$\mathcal{L}_1$ (3D volume)
Global head	$\mathcal{L}_s$ (1D global vector)	$\mathcal{L}_s$ (1D global vector)

Figure 5. Correspondence Aware Transform (CAT) generates the next feature $Z_{k+1}^{q,tr}$ via warping using a pixel-wise correspondence map, while Action Aware Transform (AAT) predicts the next feature $Z_{k+1}^{q,pr}$ using the current action a_k. While $\mathcal{L}_1$ loss encodes local information as it operates in pixel units of 3D volume, $\mathcal{L}_s$ loss encodes global information as it operates after converting 3D volume into a global vector.

The proposed method is named 'Paired Similarity' as it encodes both local and global information of agent observations. First, we generate pseudo *future frame* in two ways: Correspondence Aware Transform (CAT) and Action Aware Transform (AAT). Then, we impose similarity constraints on the target frame with the two generated *future* frames in two ways: **local** ($\mathcal{L}_1$) and **global** ($\mathcal{L}_s$) in Eq. (6), as depicted on the Figure 5.

4.2. Action Aware Transform

An episode of RL consists of states and actions in between. When a certain state is given as an input, our two-layer action aware transform module predicts the next state through a corresponding action vector a_k as a medium. Such a state prediction module has been used in the field of RL [2, 3, 7, 10], and the proposed work leverages the concept to learn effective representation using paired similarity.

References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. 3, 4

[2] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018. 5

[3] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017. 5

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015. 1

[5] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019. 1, 4, 5

[6] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 5639–5650, 2020. 1, 4

[7] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 5

[8] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, 2013. 1

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019. 2

[10] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations.

In *International Conference on Learning Representations (ICLR)*, 2021. 1, 4, 5

[11] Adam Stooke and Pieter Abbeel. rlpyt: A research code base for deep reinforcement learning in pytorch. *arXiv preprint arXiv:1909.01500*, 2019. 2

[12] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 1, 3, 5

[13] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. 2

[14] Hado P Van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019. 1