A Reinforcement Learning Background

We focus on reinforcement learning in a Markov decision process (MDP). An agent interacts with an environment, taking actions a and observing states s and rewards r. Interactions are broken up into *episodes*, series of states, actions and rewards that ultimately terminate. We denote the t-th state, action and reward as s_t, a_t and r_t respectively, following the convention that r_t is the reward received by the agent after taking action a_t in state s_t. In the control problem examined here, the agent seeks to maximize a discounted sum of rewards $G_t = \sum_{i \geq t} \gamma^i r_i$, where $\gamma \in [0, 1]$ is a discount factor balancing current and future rewards (Sutton & Barto, 2018).

Deep Q-Learning One common approach to this problem is to estimate $Q(s_t, a) = \mathbf{E}_{\pi^*}[G_t | a_t = a]$. Assuming that Q is known exactly, the problem of acting optimally is reduced to finding $\max_a Q(s_t, a)$ in each state s_t the agent encounters, which is trivial in environments that possess only a small number of possible actions. In practice, the true Q can be iteratively approximated by a parameterized Q_θ with a semi-gradient method, minimizing

$$\mathcal{L}_\theta^{DQN} = (r_t + \gamma \max_a Q_\xi(s_{t+1}, a) - Q_\theta(s_t, a_t))^2 \tag{2}$$

where Q_ξ denotes an older version of Q_θ. This method has proven extremely successful when used with deep learning, a setting referred to as Deep Q-Networks (DQN) (Mnih et al., 2015), and more broadly is a common class of deep reinforcement learning (DRL) algorithms.

A number of variants of DQN have been proposed, including those that predict full distributions of future rewards (Bellemare et al., 2017), modifications to the $\max$ operation to reduce value overestimation (Van Hasselt et al., 2016), and architectural modifications to how Q_θ is predicted (Wang et al., 2016). We employ a somewhat modified (Schwarzer et al., 2021) version of Rainbow (Hessel et al., 2018), an algorithm that combines many of these innovations.

B Uncertainty-aware comparisons

Concurrent work (Agarwal et al., 2021) has found that many prior comparisons in deep reinforcement learning are not robust and may be entirely incorrect, particularly in the Atari 100K setting. They demonstrate that these misleading comparisons are partially due to undesirable properties of the per-game median and mean normalized scores, the most commonly-used aggregate metrics, and propose using the inter-quartile mean (IQM) normalized score, calculated over *runs* rather than *tasks*. Moreover, they suggest providing percentile bootstrap confidence intervals to quantify uncertainty, to avoid misleading comparisons based on highly-variable point estimates.

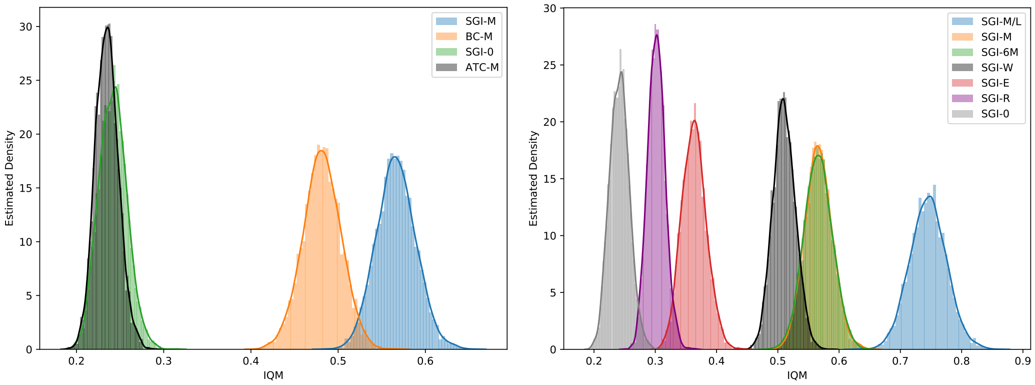
As raw per-run data is required for this, which was not reported for prior work, we do so only for experiments conducted ourselves. In the interests of improving practices in the community moving forward, we also commit to making this data for our experiments available to other researchers in the future.

In Figure 5a through Figure 5e we show estimated uncertainty via bootstrapping for the various comparisons drawn throughout Section 5, while Table 6 gives IQM human-normalized scores and 95% bootstrap confidence intervals for the same results. All comparisons in Figure 5a through Figure 5e are statistically significant ($p < 0.05$) except for:

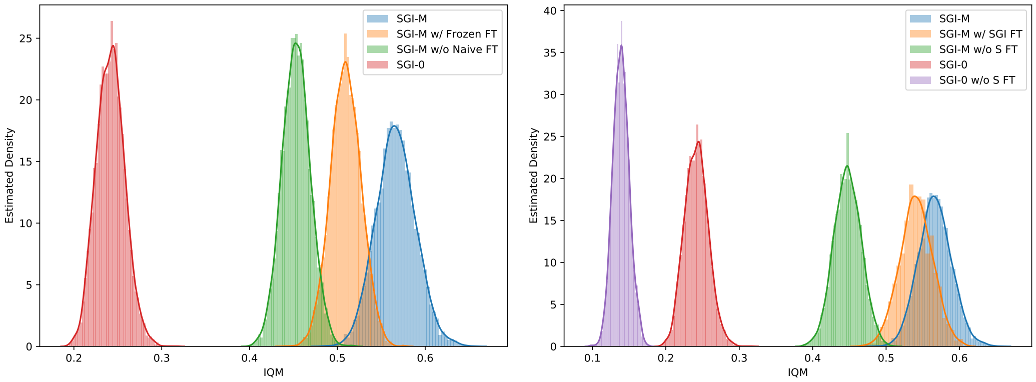
- ATC-M vs SGI-None in Figure 5a ($p \gg 0.05$)
- SGI-M vs SGI-W in Figure 5b ($p \approx 0.05$)
- SGI-M vs SGI-M w/ SGI FT in Figure 5d ($p \approx 0.4$)
- SGI-M vs G+I and S+I in Figure 5e ($p \approx 0.1$)

C Implementation Details

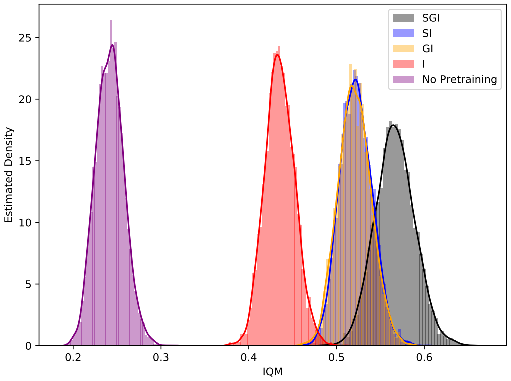
We base our work on the code released for SPR (Schwarzer et al., 2021), which in turn is based on rlpyt (Stooke & Abbeel, 2019), and makes use of NumPy (Harris et al., 2020) and PyTorch (Paszke et al., 2019).



(a) Comparisons to behavioral cloning (BC) and ATC. (b) Ablations over different pretraining datasets.



(c) Ablations over various fine-tuning. (d) Ablations over SSL objectives during fine-tuning.



(e) Ablations over pretraining SSL objectives.

Figure 5: Bootstrapping distributions for uncertainty in IQM measurements.

Table 6: Interquartile mean, median and mean human-normalized scores for variants of SGI and controls, evaluated after finetuning over all 10 runs for each of the 26 Atari 100k games. Confidence intervals computed by percentile bootstrap with 5000 resamples.

Method	IQM	95% CI	Median	95% CI	Mean	95% CI
SGI-M/L	**0.745**	(0.687, 0.805)	**0.753**	(0.625, 0.850)	**1.598**	(1.486, 1.676)
SGI-M	0.567	(0.524, 0.612)	0.679	(0.473, 0.739)	1.149	(0.974, 1.347)
SGI-M/S	0.444	(0.404, 0.487)	0.423	(0.341, 0.577)	0.914	(0.822, 1.031)
SGI-W	0.510	(0.476, 0.547)	0.589	(0.434, 0.675)	1.144	(0.981, 1.345)
SGI-E	0.363	(0.326, 0.404)	0.456	(0.309, 0.482)	0.838	(0.692, 1.008)
SGI-R	0.302	(0.275, 0.331)	0.326	(0.253, 0.385)	0.888	(0.776, 1.004)
SGI-None	0.242	(0.212, 0.274)	0.343	(0.268, 0.401)	0.565	(0.440, 0.711)
Baselines						
ATC-M	0.235	(0.210, 0.262)	0.204	(0.182, 0.291)	0.780	(0.601, 0.971)
ATC-W	0.221	(0.199, 0.244)	0.219	(0.170, 0.290)	0.587	(0.504, 0.673)
ATC-E	0.214	(0.193, 0.236)	0.237	(0.169, 0.266)	0.462	(0.420, 0.504)
ATC-R	0.187	(0.174, 0.202)	0.191	(0.139, 0.202)	0.472	(0.454, 0.491)
BC-M	0.481	(0.438, 0.524)	0.548	(0.390, 0.685)	0.858	(0.795, 0.924)
Pretraining Ablations						
S+I	0.522	(0.488, 0.559)	0.629	(0.494, 0.664)	0.978	(0.900, 1.061)
G+I	0.521	(0.486, 0.558)	0.512	(0.386, 0.582)	1.004	(0.892, 1.129)
S+G	0.032	(0.027, 0.039)	0.029	(0.025, 0.044)	0.098	(0.061, 0.146)
I	0.435	(0.404, 0.470)	0.411	(0.334, 0.489)	0.943	(0.783, 1.126)
G	0.060	(0.048, 0.072)	0.060	(0.037, 0.081)	0.181	(0.145, 0.218)
S	0.007	(0.002, 0.011)	0.009	(0.002, 0.014)	-0.054	(-0.082, -0.026)
Finetuning Ablations						
SGI-M (No S)	0.448	(0.412, 0.484)	0.419	(0.335, 0.524)	1.114	(0.921, 1.321)
SGI-None (No S)	0.139	(0.118, 0.162)	0.161	(0.123, 0.225)	0.315	(0.274, 0.356)
SGI-M (All SGI)	0.541	(0.498, 0.585)	0.397	(0.330, 0.503)	1.011	(0.909, 1.071)
SGI-M (Frozen)	0.510	(0.476, 0.543)	0.499	(0.406, 0.554)	0.971	(0.871, 1.088)
SGI-M (Naive)	0.453	(0.422, 0.485)	0.429	(0.380, 0.500)	0.845	(0.754, 0.952)

C.1 Training

We set $\lambda^{\text{SPR}} = 2$ and $\lambda^{\text{IM}} = 1$ during pre-training. Unless otherwise noted, all settings match SPR during fine-tuning, including batch size, replay ratio, target network update period, and λ^{SPR}. We use a batch size of 256 during pre-training to maximize throughput, and update both the SPR and goal-conditioned RL target network target networks with an exponential moving average with $\tau = 0.99$. We pre-train for a number of gradient steps equivalent to 10 epochs over 6M samples, no matter the amount of data used. Due to the cost of pretraining, we pre-train a single encoder per game for each configuration tested. However, we use 10 random seeds at fine-tuning time, allowing us to average over variance due to exploration and data order. Finally, we reduce fine-tuning learning rates for pretrained encoders and dynamics models by a factor of 100, and by a factor of 3 for other pretrained weights. We find this crucial to SGI's performance, and discuss it in detail in Section 5.4.

We trained SGI on standard GPUs, including V100s and P100s. We found that pretraining took roughly one to three days and finetuning between four and 12 hours per run on a single GPU, depending on the size of the network used and type of GPU.

C.2 Goal-Conditioned Reinforcement Learning

We generate goals in a three-stage process: a goal g for state s_t is initially chosen to be the target representation of a state sampled uniformly from the near future, $g \leftarrow \tilde{z}_{t+i}, i \sim \text{Uniform}(50)$, before being combined with a normalized vector of isotropic Gaussian noise n as $g \leftarrow \alpha n + (1-\alpha)g$, where $\alpha \sim \text{Uniform}(0, 0.5)$. Finally, we exchange goal vectors between states in the minibatch with probability 0.2, to ensure that some goals correspond to states reached in entirely different trajectories.

In defining our synthetic goal-conditioned rewards, we take inspiration from potential-based reward shaping (Ng et al., 1999). Using the target representations $\tilde{z}_t \triangleq f_m(s_t)$ and $\tilde{z}_{t+1} \triangleq f_m(s_{t+1})$, we define the reward as follows:

$$R(\tilde{z}_t, \tilde{z}_{t+1}, g) = d(\tilde{z}_t, g) - d(\tilde{z}_{t+1}, g) \tag{3}$$

$$d(\tilde{z}_t, g) = \exp\left(2\frac{\tilde{z}_t \cdot g}{||\tilde{z}||_2 \cdot ||g||_2} - 2\right). \tag{4}$$

As this reward function depends on the target encoder f_m, it changes throughout training, although using the slower-moving f_m rather than the online encoder f_o may provide some measure of stability. Like SPR, however, this objective is technically vulnerable to collapse. If all representations $\tilde{z}_t$ collapse to a single constant vector then all rewards will be 0, allowing the task to be trivially solved.

We estimate $Q(s_t, a_t, g)$ using FiLM (Perez et al., 2018) to condition the DQN on the goal g, which we found to be more robust than simple concatenation. A FiLM generator j produces per-channel biases β_c and scales γ_c, which then modulate features through a per-channel affine transformation:

$$\text{FiLM}(F_c|\gamma_c, \beta_c) = \gamma_c F_c + \beta_c \tag{5}$$

We use these parameters to replace the learned per-channel affine transformation in a layer norm layer (Ba et al., 2016), which we insert immediately prior to the final linear layer in the DQN head.

We apply FiLM after the first layer in the DQN's MLP head. We parameterize our FiLM generator j as a small convolutional network, which takes the goal g (viewed as a $64 \times 7 \times 7$ spatial feature map) as input and applies two 128-channel convolutions followed by a flatten and linear layer to produce the FiLM parameters γ and β.

C.3 Model Architectures

In addition to the standard three-layer CNN encoder introduced by Mnih et al. (2015), we experiment with larger residual networks (He et al., 2016). We use the design proposed by Espeholt et al. (2018) as a starting point, while still adopting innovations used in more modern architectures such as EfficientNets (Tan & Le, 2019) and MobileNetv2 (Sandler et al., 2018). In particular, we use inverted residual blocks with an expansion ratio of 2, and batch normalization (Ioffe & Szegedy, 2015) after each convolutional layer. We use three groups of three residual blocks with 32, 64 and 64 channels each, downscaling by a factor of three in the first group and two in each successive group. This yields a final representation of shape $64 \times 7 \times 7$ when applied to 84×84-dimensional Atari frames, identical to that of the standard CNN encoder. In our scaling experiment with a larger network, we increase to five blocks per group, with 48, 96 and 96 channels in each group, as well as using a larger expansion ratio of 4, producing a representation of shape $96 \times 7 \times 7$. This enlargement increases the number of parameters by roughly a factor of 5. Finally, our DQN head has 512 hidden units, as opposed to 256 in SPR.

C.4 Image Augmentation

We use the same image augmentations as used in SPR (Schwarzer et al., 2021), which itself used the augmentations used in DrQ (Kostrikov et al., 2021), in all experiments, including during both pretraining and fine-tuning. Specifically, we employ random crops (4 pixel padding and 84x84 crops) in combination with image intensity jittering.

C.5 Temporal Augmentation

Initially due to a mismatch between the semantics of the DQN Replay dataset and the buffers used by rlpyt, SGI sampled actions in the offline dataset with a temporal offset of one (i.e., sampled a_{t+1} instead of a_t). Surprisingly, we found that fixing this did not improve performance as would be expected. Upon further inspection, the actions sampled were identical to the true actions a large fraction of the time, but sampling the true actions made SGI's pretraining tasks far easier. In other words, this was effectively serving as an additional form of data augmentation.

Table 7: Performance of SGI with various action offsets. We report results with offset 1 in the main paper.

Method	IQM@0	IQM@1	Median@0	Median@1	Mean@0	Mean@1
SGI-M	0.530	0.567	0.490	0.679	1.120	1.149
SGI-W	0.553	0.510	0.463	0.589	1.229	1.144

As we find that this in fact improves performance on the median game, the traditional focus of the community, we continue to report results with the original offset. However, under the evaluation method proposed by (Agarwal et al., 2021) these methods are roughly equivalent (see Table 7), and we suspect that methods seeking to build off of SGI for offline RL or imitation learning will wish to sample actions without an offset. We have therefore made this configurable in our released code.

C.6 Experiments with ATC

As ATC (Stooke et al., 2021) was not tested on the Atari100k setting, and as its hyperparameters (including network size and fine-tuning scheme) are very different from those used by SGI, we modify its code[6] to allow it to be fairly compared to SGI. We replace the convolutional encoder with that used by SGI, and use the same optimizer settings, image augmentation, pre-training data, and number of pre-training epochs as in SGI. However, we retain ATC's mini-batch structure (i.e., sampling 32 subsequences of eight consecutive time steps, for a total batch size of 512), as this structure defines the negative samples used by ATC's InfoNCE loss. During fine-tuning, we transfer the ATC projection head to the first layer of the DQN MLP head, as in SPR; we otherwise fine-tune identically to SGI, including using SPR.

---

[6] `https://github.com/astooke/rlpyt/tree/master/rlpyt/ul`

D Pseudocode

Algorithm 1: Pre-Training with SGI

```
Denote parameters of online encoder $f_o$, projection $p_o$ and Q-learning head as $\theta_o$;
Denote parameters of target encoder $f_m$, projection $p_m$ and Q-learning target head as $\theta_m$;
Denote parameters of transition model $h$, predictor $q$, inverse model $I$ as $\phi$;
Denote the maximum prediction depth as $K$, batch size as $N$;
Denote distance function in goal RL reward as $d$;
initialize offline dataset $D$;
while Training do
    sample a minibatch of sequences of $(s_t, a, s+t+1) \sim D$ ; // sample unlabeled data
    /* sample goals */
    for $i$ in range$(0, N)$ do
        $s^i \leftarrow$ augment$(s^i)$; ${s'}^i \leftarrow$ augment$({s'}^i)$ ; // augment input images
        $j \sim$ Discrete Uniform$(1, 50)$ ; // sample hindsight goal states
        $g^i \leftarrow f_m(s_j^n)$ ; // encode goal states
        $\alpha \sim$ Uniform$(0, 0.5)$, $n \sim$ Normal$(0, 1)$ ; // sample noise parameters
        $g^i \leftarrow \alpha g^i + (1 - \alpha) n$ ; // apply noise
        /* Permute to make some goals very challenging to reach */
        permute $\sim$ Bernoulli$(0.2)$
        if permute then
            $j \sim$ Discrete Uniform$(N)$
            $g^i \leftarrow g^j$ ; // permute goal
    /* compute SGI loss */
    for $i$ in range$(0, N)$ do
        $\hat{z}_0^i \leftarrow f_\theta(s_0^i)$ ; // compute online representations
        $l^i \leftarrow 0$;
        /* compute SPR loss */
        for k in (1, ..., K) do
            $\hat{z}_k^i \leftarrow h(\hat{z}_{k-1}^i, a_{k-1}^i)$ ; // latent states via transition model
            $\tilde{z}_k^i \leftarrow f_m(s_k^i)$ ; // target representations
            $\hat{y}_k^i \leftarrow q(p_o(\hat{z}_k^i))$, $\tilde{y}_k^i \leftarrow g_m(\tilde{z}_k^i)$ ; // projections
            $l^i \leftarrow l^i - \lambda^{\text{SPR}} \left(\frac{\tilde{y}_k^i}{||\tilde{y}_k^i||_2}\right)^\top \left(\frac{\hat{y}_k^i}{||\hat{y}_k^i||_2}\right)$ ; // SGI loss at step $k$
        /* compute inverse modeling loss */
        for k in (1, ..., K) do
            $l^i \leftarrow \lambda^{\text{IM}} \cdot$ Cross-entropy loss$(a_{k-1}^i, I(\hat{y}_{k-1}, \tilde{y}_k))$
        /* compute goal RL loss */
        $r^i \leftarrow d(g^i, \tilde{z}_t) - d(g^i, \tilde{z}_{t+1})$ ; // Calculate goal RL reward
        $l^i \leftarrow l^i +$ RL loss$(s^i, a^i, r^i, {s'}^i)$ ; // Add goal RL loss for batch
    $l \leftarrow \frac{1}{N} \sum_{i=0}^{N} l^i$ ; // average loss over minibatch
    $\theta_o, \phi \leftarrow$ optimize$((\theta_o, \phi), l)$ ; // update online parameters
    $\theta_m \leftarrow \tau \theta_o + (1 - \tau) \theta_m$ ; // update target parameters
return $(\theta_o,\ \phi)$ ; // return parameters for fine-tuning
```

E Full Results on Atari100k

We report full scores for SGI agents across all 26 games in Table 8. We do not reproduce the per-game scores for APT and VISR provided by Liu & Abbeel (2021), as we believe that the scores in the currently-available version of their paper may contain errors.[7]

Table 8: Mean return per episode for the 26 Atari100k games (Kaiser et al., 2019) after 100k steps. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds. We reproduce scores for SPR from Schwarzer et al. (2021), whereas ATC scores are from our implementation.

	Random	Human	SPR	ATC-M	SGI-R	SGI-E	SGI-W	SGI-M/S	SGI-M	SGI-M/L
Alien	227.8	7127.7	801.5	699.0	1034.5	857.6	1043.8	1070.5	1101.7	**1184.0**
Amidar	5.8	1719.5	176.3	95.4	154.8	166.8	**206.7**	185.9	168.2	171.2
Assault	222.4	742.0	571.0	509.8	446.6	583.1	759.5	632.4	905.1	**1326.5**
Asterix	210.0	8503.3	977.8	454.1	754.6	953.6	**1539.1**	651.8	835.6	567.2
Bank Heist	14.2	753.1	380.9	534.9	397.4	514.8	426.3	547.4	**608.4**	567.8
Battle Zone	2360.0	37187.5	**16651.0**	13683.8	4439.0	16417.0	7103.0	12107.0	13170.0	14462.0
Boxing	0.1	12.1	35.8	16.8	57.7	33.6	50.2	40.0	36.9	**73.9**
Breakout	1.7	30.5	17.1	16.9	23.4	17.8	35.4	23.8	42.8	**251.9**
Chopper Command	811.0	7387.8	974.8	870.8	784.7	1136.2	1040.1	1042.7	**1404.0**	1037.9
Crazy Climber	10780.5	35829.4	42923.6	74215.5	50561.2	76356.3	81057.4	75542.1	88561.2	**94602.2**
Demon Attack	152.1	1971.0	545.2	524.6	2198.7	357.5	1408.5	1135.5	968.1	**5634.8**
Freeway	0.0	29.6	24.4	5.7	2.1	15.1	26.5	12.5	**30.0**	28.6
Frostbite	65.2	4334.7	**1821.5**	222.6	349.3	981.4	247.7	861.1	741.3	927.8
Gopher	257.6	2412.5	715.2	946.2	1033.9	964.9	1846.0	1172.4	1660.4	**2035.8**
Hero	1027.0	30826.4	7019.2	6119.4	7875.2	6863.7	7503.9	7090.4	7474.0	**9975.9**
Jamesbond	29.0	302.8	365.4	272.6	263.9	383.8	**425.1**	413.2	366.4	394.8
Kangaroo	52.0	3035.0	**3276.4**	603.1	923.8	1588.9	598.6	1236.8	2172.8	1887.5
Krull	1598.0	2665.5	3688.9	4494.7	5672.6	4070.7	5583.2	**6161.3**	5734.0	5862.6
Kung Fu Master	258.5	22736.3	13192.7	11648.2	13349.2	11802.1	14199.7	16781.8	16137.8	**17340.7**
Ms Pacman	307.3	6951.6	1313.2	848.9	411.0	1278.3	1970.8	1519.5	1520.0	**2218.0**
Pong	-20.7	14.6	-5.9	-13.5	-3.9	4.2	4.7	**9.7**	7.6	7.7
Private Eye	24.9	69571.3	**124.0**	95.0	95.3	100.0	100.0	84.7	90.0	83.8
Qbert	163.9	13455.0	669.1	572.2	595.0	717.6	**855.6**	804.7	709.8	702.6
Road Runner	11.5	7845.0	14220.5	7989.3	5476.0	9195.2	18011.9	12083.5	**18370.2**	18306.8
Seaquest	68.4	42054.7	583.1	415.7	735.3	615.2	656.1	728.2	728.4	**1979.3**
Up N Down	533.4	11693.2	28138.5	84361.2	67968.1	63612.9	**84551.4**	42165.6	79228.8	46083.3
Median HNS	0.000	1.000	0.415	0.204	0.326	0.456	0.589	0.423	0.679	**0.755**
Mean HNS	0.000	1.000	0.704	0.780	0.888	0.838	1.144	0.914	1.149	**1.590**
#Games > Human	0	0	7	5	5	6	8	6	**9**	**9**
#Games > 0	0	26	**26**	**26**	25	**26**	**26**	**26**	**26**	**26**

---

[7] In particular, we observed that VISR claimed to have a score below -21 on Pong, which is impossible with standard settings.

Table 9: Mean return per episode for the 26 Atari100k games (Kaiser et al., 2019) after 100k steps for versions of SGI with modified fine-tuning, as discussed in Section 5. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds. We reproduce scores for SPR from Schwarzer et al. (2021).

	Random	Human	SGI-None	Naive	Frozen	No SPR	Full SSL	SGI-M
Alien	227.8	7127.7	835.9	1049.3	**1242.8**	1060.7	1117.6	1101.7
Amidar	5.8	1719.5	107.6	133.6	147.7	154.2	**206.0**	168.2
Assault	222.4	742.0	657.7	752.1	869.2	756.3	**1145.2**	905.1
Asterix	210.0	8503.3	832.9	**1029.3**	433.1	575.5	603.1	835.6
Bank Heist	14.2	753.1	613.2	**726.5**	273.6	365.8	323.4	608.4
Battle Zone	2360.0	37187.5	13490.0	**15708.0**	11754.0	13692.0	11689.8	13170.0
Boxing	0.1	12.1	6.6	24.0	**61.5**	34.7	42.7	36.9
Breakout	1.7	30.5	12.1	29.3	34.0	43.0	**62.6**	42.8
Chopper Command	811.0	7387.8	1085.2	1081.2	916.5	925.5	965.8	**1404.0**
Crazy Climber	10780.5	35829.4	19707.6	55002.4	65220.0	69505.6	69052.0	**88561.2**
Demon Attack	152.1	1971.0	778.8	850.0	1329.4	981.7	**1783.8**	968.1
Freeway	0.0	29.6	17.2	28.1	24.4	13.2	10.9	**30.0**
Frostbite	65.2	4334.7	1475.8	662.1	1045.4	482.1	**1664.9**	741.3
Gopher	257.6	2412.5	438.2	626.1	**2214.1**	1561.7	1998.7	1660.4
Hero	1027.0	30826.4	6472.0	5538.3	6353.3	5249.6	**8715.4**	7474.0
Jamesbond	29.0	302.8	157.4	324.2	358.2	346.8	**407.6**	366.4
Kangaroo	52.0	3035.0	**3802.8**	3091.6	800.0	685.6	999.5	2172.8
Krull	1598.0	2665.5	3954.0	5202.7	**6073.7**	5722.8	5323.9	5734.0
Kung Fu Master	258.5	22736.3	7929.4	11952.2	**19374.6**	15039.8	18123.2	16137.8
Ms Pacman	307.3	6951.6	990.2	1276.4	1663.3	1753.3	**1779.3**	1520.0
Pong	-20.7	14.6	-4.4	-4.2	3.8	3.9	-0.1	**7.6**
Private Eye	24.9	69571.3	62.8	**385.9**	96.7	90.5	90.0	90.0
Qbert	163.9	13455.0	720.0	664.8	587.6	681.3	**3015.8**	709.8
Road Runner	11.5	7845.0	5428.4	14629.7	14311.9	17036.5	13998.2	**18370.2**
Seaquest	68.4	42054.7	577.8	509.0	1054.4	**1397.8**	989.4	728.4
Up N Down	533.4	11693.2	46042.6	48856.6	29938.4	**105466.9**	45023.5	79228.8
Median HNS	0.000	1.000	0.343	0.425	0.499	0.452	0.397	**0.679**
Mean HNS	0.000	1.000	0.565	0.849	0.971	1.114	1.011	**1.149**
#Games > Human	0	0	3	8	8	8	8	**9**
#Games > SPR	0	19	10	14	15	14	17	**20**

Table 10: Mean return per episode for the 26 Atari100k games (Kaiser et al., 2019) after 100k steps for various combinations of SGI's pretraining objectives, as discussed in Section 5. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds.

	Random	Human	None	S	G	I	G+I	S+G	S+I	SGI
Alien	227.8	7127.7	835.9	278.7	964.3	1161.6	571.2	1172.3	**1203.0**	1101.7
Amidar	5.8	1719.5	107.6	37.8	54.8	198.1	58.0	**210.5**	175.4	168.2
Assault	222.4	742.0	657.7	517.9	512.3	868.1	567.2	813.5	820.3	**905.1**
Asterix	210.0	8503.3	832.9	292.6	416.1	475.6	431.8	506.3	648.5	**835.6**
Bank Heist	14.2	753.1	**613.2**	3.1	115.2	357.6	57.2	423.3	547.5	608.4
Battle Zone	2360.0	37187.5	13490.0	4665.0	3336.0	14807.0	3249.0	12528.0	**15491.0**	13170.0
Boxing	0.1	12.1	6.6	-21.8	12.5	40.1	-0.4	**42.9**	38.3	36.9
Breakout	1.7	30.5	12.1	0.9	2.1	24.1	3.2	41.0	41.6	**42.8**
Chopper Command	811.0	7387.8	1085.2	799.7	813.1	973.1	923.7	1097.2	978.3	**1404.0**
Crazy Climber	10780.5	35829.4	19707.6	243.3	17760.3	51203.9	581.0	66228.5	83995.4	**88561.2**
Demon Attack	152.1	1971.0	778.8	668.9	316.9	**1524.6**	756.4	1008.4	1286.6	968.1
Freeway	0.0	29.6	17.2	15.2	17.7	2.6	19.3	**30.5**	29.1	30.0
Frostbite	65.2	4334.7	**1475.8**	427.2	523.3	395.0	215.4	530.5	463.3	741.3
Gopher	257.6	2412.5	438.2	60.7	129.0	**1966.1**	99.0	1747.4	1778.7	1660.4
Hero	1027.0	30826.4	6472.0	2381.2	3590.2	7177.6	3998.7	**8251.2**	7366.2	7474.0
Jamesbond	29.0	302.8	157.4	41.8	236.0	373.1	183.6	365.6	**378.4**	366.4
Kangaroo	52.0	3035.0	**3802.8**	129.8	401.6	1041.4	222.6	830.8	760.2	2172.8
Krull	1598.0	2665.5	3954.0	720.1	1241.4	**5859.8**	1582.4	5778.8	5808.6	5734.0
Kung Fu Master	258.5	22736.3	7929.4	79.7	453.7	16914.7	686.2	**17825.1**	14681.9	16137.8
Ms Pacman	307.3	6951.6	990.2	418.7	528.5	1620.1	293.3	**1847.1**	1715.9	1520.0
Pong	-20.7	14.6	-4.4	-20.9	-20.4	-3.0	-21.0	0.9	1.7	**7.6**
Private Eye	24.9	69571.3	62.8	-20.7	89.4	**100.0**	12.7	98.2	**100.0**	90.0
Qbert	163.9	13455.0	**720.0**	201.0	277.4	706.5	215.2	650.5	601.9	709.8
Road Runner	11.5	7845.0	5428.4	780.3	5592.9	17698.4	2617.8	18229.4	17443.5	**18370.2**
Seaquest	68.4	42054.7	577.8	105.7	193.2	965.3	118.8	**1115.0**	792.1	728.4
Up N Down	533.4	11693.2	46042.6	892.2	4399.7	58142.0	1313.4	52772.9	39771.3	**79228.8**
Median HNS	0.000	1.000	0.343	0.009	0.060	0.411	0.029	0.512	0.629	**0.679**
Mean HNS	0.000	1.000	0.565	-0.054	0.181	0.943	0.098	1.004	0.978	**1.149**
#Games > Human	0	0	3	0	1	7	0	**9**	8	**9**
#Games > SPR	0	19	10	1	1	18	1	**20**	19	**20**

F Transferring Representations between Games

One advantage of pretraining representations is the possibility of representations being useful across games. Intuitively, we expect better transfer between similar games so we chose five "cliques" of games with similar semantics and visual elements. The cliques are shown in Table 11. We pretrain on a dataset of 750k frames from each game in a clique (i.e. 3M frames for a clique of 4) and finetune on a single game. To show whether pretraining on other games is beneficial, we compare to a baseline of pretraining on just the 750k frames from the single Atari 100k game we use for finetuning.

Our results in Table 12 show that pretraining with the extra frames from the clique games is mostly unhelpful to finetune performance. Only Kangaroo shows a modest improvement, a few games show no difference in performance, and most games show a decrease in performance when pretraining with other games. We believe that Atari may not be as suitable to transferring representations as other domains, and previous work using Atari to learn transferable representations has also had negative results (Stooke et al., 2021). Though game semantics can be similar, we note that even small differences in rule sets and visual cues can make transfer difficult.

Table 11: Cliques of semantically similar games

Clique	Games
space	Space Invaders, Assault, Demon Attack, Phoenix
pacman	MsPacman, Alien, Bank Heist, Wizard Of Wor
platformer	Montezuma Revenge, Hero, Kangaroo, Tutankham
top scroller	Crazy Climber, Up N Down, Skiing, Journey Escape
side scroller	Chopper Command, James Bond, Kung Fu Master, Private Eye

Table 12: Mean return per episode for clique games in Atari100k (Kaiser et al., 2019) after 100k steps. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds. Games in the same clique are placed together.

Game	Single	Clique
Assault	738.5	554.1
Demon Attack	1171.8	695.0
Alien	1183.9	830.2
Bank Heist	448.8	303.0
Ms Pacman	1595.8	1352.1
Kangaroo	489.2	994.0
Crazy Climber	52036.0	21829.8
Up N Down	18974.7	13493.9
James Bond	397.6	325.4
Kung Fu Master	16402.6	16499.0
Chopper Command	933.6	854.6