We would like to thank all reviewers; the comments are very helpful and we believe that we have addressed the concerns. The modifications have led to a much improved paper. Below we provide a point-by-point response.

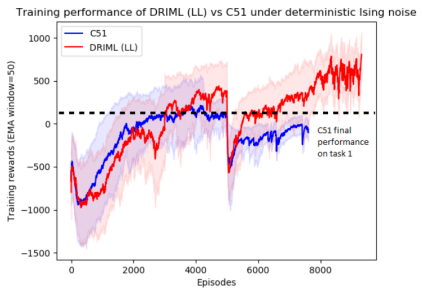
Additional result: Adaptive DRIML In our original submission, we fixed the temporal offset k of DRIML and found that performance varied across games (see Procgen results in Table 4). This supports a hypothesis that different games express their dynamics at different time scales, so it might be important to look farther ahead (larger k) in DRIML for slower games. We learn an "adaptive" k (dependent on the game) directly from the RL policy, using a simple single-hidden-layer neural network trained with SGD that predicts the next action conditioned on the previous action given the current policy. During representation learning, given the sequence of actions from the same trajectory we are drawing states (i.e., the buffer), we sample the length of the sequence of actions k under this transition model by sequential sampling as a Markov process. As can be seen from our results below, DRIML-ada outperforms DRIML-fixed, achieving nearly double the average score of C51. While this modification is small in terms of the auxiliary loss (it learns k without modifying the core algorithm), this addition is novel, insightful, and useful to performance.

[Differences with and comparison to CURL and CPC] DRIML encourages consistency in state representations across multiple time steps conditioned on the action through a contrastive loss between pairs of global and local (corresponding to a patches) state representations at different time steps (offset by k time steps). CURL only does contrastive learning between augmented copies of the *same* global state, and does so by relying only on data augmentation and without action-conditioning. We tested data augmentation a la CURL on DRIML (results in Table 3), on some games perf improved, not so much on others. Like DRIML, CPC also predicts future states, but it does not use action-conditioning and relies on autoregression modeled by an RNN. We include additional results below, which are from the best agents on 3 seeds over 50M timesteps: our implementation of 5-step CPC; DRIML-fix and DRIML-ada; CURL (using their code); and DRIML-noact (without actions). All baselines were tuned with the same computation budget. The last row shows the cumulative score over all 16 games normalized by max-min performance as in Cobbe et al. (2019), and wrt C51. We conclude that the additions from DRIML are essential to achieving good performance on ProcGen, as DRIML is the only method that significantly outperforms C51 alone.

[PacMan + Ising] As suggested by R1, we also ran DRIML-fix and C51 on our simple PacMan game augmented with an Ising model in irrelevant parts of the screen (e.g. walls). The figure below shows the training returns over 8,000 episodes. This further supports already-established conclusions from the paper.

[DRIML and DeepMDP] DeepMDP is an interesting representation-learning method that learns latent reward and transition models from a bisimulation perspective. An important property of maximal bisimulations is that states from which the optimal action sequences are identical are merged. This will make the DeepMDP encoder ignore all state information irrelevant to the optimal policy. In our PacMan experiment, the embedding will not include any knowledge about the harmless ghosts (as they do not impact the actions to take). Upon change of the harmful ghost's color, the encoder will thus need extra samples to adapt. We will expand our discussion of DeepMDP.

[Other comments] 1) We report performance on Procgen instead of Atari / MuJoCo as Procgen allows for extremely fast procedural generation of multiple levels, and is better suited for testing continual learning/multi-level setups. 2) We will move their theory to the Appendix as we agree they are not an essential component of the paper. 3) The choice of $\alpha = 0.499$ for the MC experiment is to approach 0.5 (lowest MI possible) while allowing the matrix to be invertible. Fig. 2d shows that for this class of MC, the spectral gap encodes the "predictivity" of the system, and hence infoNCE is expected to converge faster on α values further from 0.5. 4) Our implementation relies on `rlpyt` (Stooke and Abbeel, 2019), parameters such as gradient clipping, n-step returns and soft updates were taken to be default values in that library. In Fig.3, the similarity scores are computed between $t = 2$ and $t = 3$, but the grayscale plots show $t = 32$ for visibility (at $t = 2$ the difference between random noise and Ising is barely visible). 5) The reviewers correctly pointed out some typos and presentation issues (e.g. font size), which will be fixed in the next version.



Env	C51	CPC-1→ 5	CURL	DRIML-noact	DRIML-fix	DRIML-ada
bigfish	1.33±0.12	1.17±0.16	2.7±1.3	1.19±0.04	2.02±0.18	**4.45±0.71**
bossfight	0.57±0.05	0.52±0.07	0.6±0.06	0.47±0.01	0.67±0.02	**1.05±0.19**
caveflyer	9.19±0.29	6.4±0.56	6.94±0.25	8.26±0.26	**10.18±0.41**	6.77±0.04
chaser	0.22±0.04	0.21±0.02	0.35±0.04	0.23±0.02	0.29±0.02	**0.38±0.04**
climber	1.68±0.1	1.71±0.11	1.75±0.09	1.57±0.01	**2.26±0.05**	2.2±0.08
coinrun	**29.7±5.44**	11.4±1.55	21.17±1.94	13.15±1.21	27.24±1.92	22.88±0.4
dodgeball	1.2±0.08	1.05±0.04	1.09±0.04	1.22±0.04	1.28±0.02	**1.44±0.06**
fruitbot	3.86±0.96	4.56±0.93	4.89±0.71	5.42±1.33	5.4±1.02	**9.53±0.29**
heist	1.54±0.1	0.93±0.08	1.06±0.05	1.04±0.02	1.3±0.05	**1.89±0.02**
jumper	**13.22±0.83**	2.28±0.44	10.27±0.61	4.31±0.64	12.64±0.64	12.16±0.42
leaper	5.03±0.14	4.01±0.71	3.94±0.46	5.4±0.09	6.17±0.29	**6.35±0.46**
maze	2.36±0.09	1.14±0.08	0.82±0.2	1.44±0.26	1.38±0.08	**2.62±0.1**
miner	0.13±0.01	0.13±0.02	0.1±0.0	0.12±0.01	0.14±0.01	**0.19±0.02**
ninja	**9.36±0.01**	6.23±0.82	5.84±1.21	6.44±0.22	9.21±0.25	8.74±0.28
plunder	2.99±0.07	3.0±0.06	2.77±0.14	3.2±0.05	3.37±0.17	**3.58±0.04**
starpilot	2.44±0.12	2.87±0.05	2.68±0.09	3.7±0.3	**4.56±0.21**	2.63±0.16
Norm.score	1.0	0.23	0.52	0.59	1.48	1.9