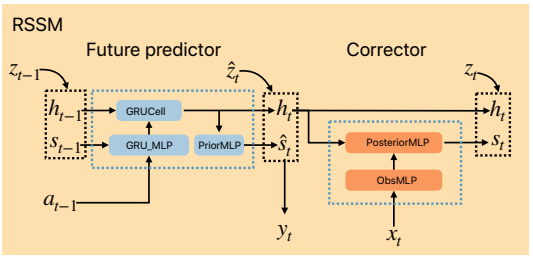
A Architecture and Training Details

In this section, we present the architecture and training hyper-parameters for the proposed CoRe model in more detail.

A.1 GRU-RNN with stochastic and deterministic states

CoRe uses a recurrent state-space model (RSSM) that is based on GRUs (Cho et al., 2014) and similar to the one used in Dreamer (Hafner et al., 2020), as shown in the figure below. Latent states at any time-step consist of a (deterministic) recurrent hidden state and a stochastic state. The prior latent state is $\hat{\boldsymbol{z}}_t = [\boldsymbol{h}_t, \hat{\boldsymbol{s}}_t]$ and the posterior latent state is $\boldsymbol{z}_t = [\boldsymbol{h}_t, \boldsymbol{s}_t]$. They share the same recurrent hidden state $\boldsymbol{h}_t$ but differ in the stochastic component. $\hat{\boldsymbol{s}}_t$ depends on $\boldsymbol{h}_t$, whereas $\boldsymbol{s}_t$ depends on $\boldsymbol{h}_t$ and the new observation features $\boldsymbol{x}_t$. The feed-forward operation is

$$
\begin{aligned}
\boldsymbol{h}_t &= \text{GRUCell}(\boldsymbol{h}_{t-1}, \text{GRU\_MLP}(\boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})), \\
\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}} &= \text{PriorMLP}(\boldsymbol{h}_t), \\
\hat{\boldsymbol{s}}_t &= \hat{\boldsymbol{\mu}} + \hat{\boldsymbol{\sigma}} * \mathcal{N}(0, 1), \\
\boldsymbol{\mu}, \boldsymbol{\sigma} &= \text{PosteriorMLP}(\boldsymbol{h}_t, \text{ObsMLP}(\boldsymbol{x}_t)), \\
\boldsymbol{s}_t &= \boldsymbol{\mu} + \boldsymbol{\sigma} * \mathcal{N}(0, 1),
\end{aligned}
$$



where MLP stands for a fully-connected multi-layered neural network. The figure above illustrates this operation. It can be seen as a more detailed view of the RSSM shown in Figure 1(b). Note that the recurrent hidden state $\boldsymbol{h}_t$ is propagated through time without sampling. This is important because sampling can make it hard to retain information for long time-scales and destroys smoothness of the latent state. However, information inserted into the RNN through $\boldsymbol{s}_t$ is stochastic, making the RSSM capable of generating multiple futures. This prevents mode-averaging and allows the model to generate diverse trajectories when doing rollouts.

A.2 Architecture

The model consists of a number of components: observation encoder CNN, RSSM (which includes PriorMLP, PosteriorMLP, GRU_MLP, GRUCell, ObsMLP, along with the reward, inverse dynamics, and observation representation decoder networks), and the actor and critic networks. Table 1 describes the architecture of these networks. CNN layers are denotes as $[\text{num\_filters}, \text{kernel\_size}, \text{stride}]$. We borrow the observation encoder architecture from previous work (Laskin et al., 2020a; Yarats et al., 2019) and use it as-is to ensure an accurate comparison. ELU non-linearity (Clevert et al., 2016) is used in all places except the observation encoder (which uses ReLU units).

A.3 Training

The training is broken into iterations, where in each iteration one episode of data are collected, added to the replay buffer, and a number of gradient steps are taken using mini-batches sampled from the replay buffer.

Data Collection For the first 1000 steps, data are collected by taking actions sampled from a uniform distribution in $[-1, 1]$. After that, data are collected by taking actions sampled from the learned actor policy, which requires rolling out the RSSM. At each training iteration, an entire episode of data are collected, where the length of the episode is 1000 steps of the underlying MuJoCo simulator. The actual number of steps is 1000 divided by the number of times the same action is repeated, which is standard based on the task (Table 3). We rollout an entire episode because we use a recurrent model and it seemed natural to use the same model through time within an episode. Typically, model-free RL algorithms collect data one environment step at a time. We did not explore per-step data collection in this work, although that could work just as well. The data from the Distracting Control Suite is rendered at a 320×240 resolution which is resized to 100×100. Pixels are normalized to $[0, 1]$ by dividing by 255. All actions are normalized to lie in $[-1, 1]$ and are modeled using tanh-squashed Gaussian distributions.

Component	Architecture
Observation Encoder	CNN : $[32, 3\times3, 2]$, $[32, 3\times3, 1]\times3$, 50 fully-connected, layer norm
PriorMLP	$[400, 400, 400]$
PosteriorMLP	$[400, 400]$
GRUCell	200 hidden units
GRU_MLP	$[400, 400]$
ObsMLP	$[256, 256]$, layer norm
Reward prediction	$[128, 128, 1]$
Inverse dynamics prediction	$[128, 128, a_{\text{dims}}]$
Observation representation decoder	$[256, 256, 50]$, layer-norm
Actor	$[1024, 1024, 1024, 2a_{\text{dims}}]$
Critic	$[1024, 1024, 1024, 1]$
Gating (optional)	CNN :$[16, 5 \times 5, 1] \times 3$, $[1, 1 \times 1, 1]$, sigmoid.

Table 1: Architecture of model components.

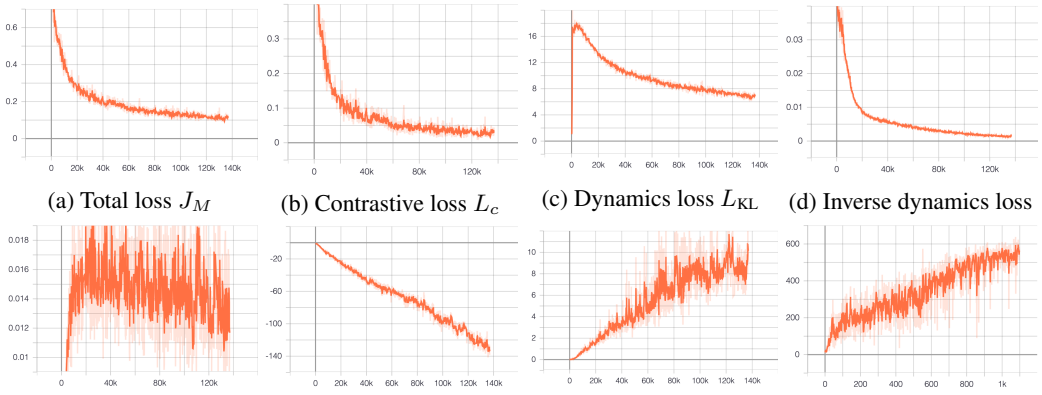
Parameter	Value
Replay Buffer	100,000
Initial steps	1000
Model Learning rate	3.e-4
Critic Learning rate	1.e-3
Actor Learning rate	1.e-3
Target entropy	-a_{dims}
Actor update frequency	1
Target critic update frequency	1
Target critic update τ	0.005
Action log std range	[-10, 2]
KL-weight β	0.01
Reward prediction weight α_r	1.0
Inverse Dynamics weight α_a	1.0
Weight decay	0
Critic max grad norm clip	100
Actor max grad norm clip	10
Model max grad norm clip	10

Table 2: List of hyper-parameters.

Model updates Training is done with mini-batches of $N = 32$ sequences of length $T = 32$ each sampled from the replay buffer. Data augmentation is done by taking random 84×84 crops. The same crop position is used across the entire sequence. Each mini-batch is used to do three updates sequentially corresponding to the three losses: J_M, J_Q and J_π. The number of updates is chosen to be 0.5 times the number of steps in an episode. All training hyper-parameters are listed in Table 2.

Implementation The model is implemented using PyTorch (Paszke et al., 2019). The environment is based on MuJoCo (Todorov et al., 2012). All training was done on single Nvidia A100 GPUs. Training time for 500K updates varies from 12-24 hr depending on the task. Training times differ due to different values of action repeat. Our implementation will be made public.

A.4 Training curves for individual loss terms



(a) Total loss J_M (b) Contrastive loss L_c (c) Dynamics loss L_{KL} (d) Inverse dynamics loss

(e) Reward loss $(r_t - \hat{r}_t)^2$ (f) Actor loss J_π. (g) Critic loss J_Q (h) Episode reward

Figure 1: Training curves for a typical run of CoRe training on cheetah, dynamic-medium setting.

There are a number of loss terms which are linearly combined to create the total model loss J_M. In this section, we present training curves that show how these individual terms optimize with training. Figure 1 shows these loss terms, along with the RL training losses (J_Q and J_π) and the episode reward. We can see that the total loss J_M goes down as expected, along with the individual terms. Note that the reward loss is low in the beginning because the agent gets zero rewards which is easy for the model to predict. As the agent starts receiving better rewards, the reward error increases but then eventually starts to come down. The KL-term has a similar behavior. At the very beginning, the prior and posterior latent state distributions match each other (making their KL divergence small)

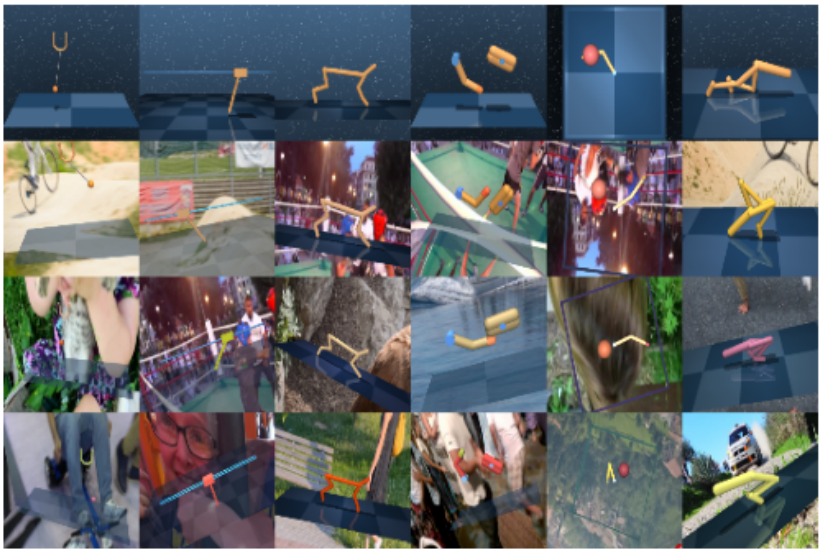


Figure 2: Examples from the Distracting Control Suite (Stone et al., 2021). The top row shows the clean versions of each of the six tasks. The subsequent rows show examples from the easy, medium, and hard distraction settings.

but both are bad at modeling the data. As more training data is encountered, the KL rises sharply, then eventually reduces with more training. The actor loss (which is dominated by the negative of the Q-value of the actor's chosen action) goes down as expected, indicating that the actor outputs actions that have high Q-values. The critic loss appears to increase because the magnitude of the Q-values increases with training, causing the magnitude of the Bellman residual to go up as well.

B Complete Results on Distracting Control Suite

In this section, we report the performance of CoRe on all the distraction settings in the Distracting Control Suite. There are three difficulty levels: easy, medium, and hard[1] as shown in Figure 2. The difficulty is set by increasing the scale of the camera pose and color change and the number of background distractions that are used (Table 4). For each difficulty level, there are two settings: **static**, in which the distraction (a background image, or a particular choice of random camera pose) is fixed throughout the episode, and **dynamic** in which the distraction changes smoothly. In the case of background distractions, the video plays back-and-forth, ensuring no sharp cuts. In the case of the camera distractions, the camera moves along a smooth trajectory.

Task	Action repeat
Ball in cup catch	4
Cartpole swingup	8
Cheetah Run	4
Finger spin	2
Reacher easy	4
Walker walk	2

Table 3: Action repeat

Difficulty	Train videos	Val videos	Camera and Color change scale
Easy	4	4	0.1
Medium	8	8	0.2
Hard	60	30	0.3

Table 4: Distraction settings in the Distracting Control Suite.

Table 5 compares the performance of CoRe with model-free RL baselines such as SAC (Haarnoja et al., 2018) and QT-Opt (Kalashnikov et al., 2018) combined with data augmentation techniques RAD (Laskin et al., 2020b) and DrQ (Yarats et al., 2021) as reported in (Stone et al., 2021). We also compare with a recurrent SAC+RAD baseline which uses the same recurrent architecture as CoRe, but

[1]The 'hard' level is described in the code but results are not reported in the main paper (Stone et al., 2021).

Table 5: Results on the Distraction Control Suite benchmark. All results reported at 500K steps, unless mentioned otherwise. **Bold** numbers indicate the results from the best performing models at 500K steps.

(a) Benchmark easy. Mean reward ± Standard Error. **Bold** indicates best results at 500K steps.

	Method	Mean	BiC-Catch	C-swingup	C-run	F-spin	R-easy	W-walk
Static	SAC+RAD	182 ± 24	129 ± 20	360 ± 25	72 ± 44	370 ± 114	102 ± 14	60 ± 31
	QT-Opt+RAD	317 ± 8	218 ± 44	446 ± 23	220 ± 5	711 ± 27	181 ± 17	128 ± 14
	QT-Opt+DrQ	299 ± 6	217 ± 35	416 ± 20	199 ± 8	695 ± 33	171 ± 25	93 ± 9
	Recurrent SAC+RAD	231 ± 23	99 ± 23	432 ± 10	312 ± 9	18 ± 16	89 ± 5	436 ± 33
	CURL	418 ± 32	165 ± 35	430 ± 30	357 ± 13	759 ± 19	142 ± 25	657 ± 47
	CoRe	**634 ± 29**	**854 ± 12**	**562 ± 15**	**459 ± 14**	**870 ± 39**	**319 ± 46**	**742 ± 30**
	CoRe (1M steps)	769 ± 18	876 ± 15	681 ± 15	596 ± 13	920 ± 32	666 ± 34	875 ± 9
Dynamic	SAC+RAD	270 ± 31	366 ± 59	297 ± 21	198 ± 39	338 ± 59	173 ± 11	249 ± 138
	QT-Opt+RAD	343 ± 24	490 ± 64	467 ± 12	170 ± 8	393 ± 91	**428 ± 68**	109 ± 12
	QT-Opt+DrQ	265 ± 5	395 ± 39	431 ± 18	126 ± 10	203 ± 33	343 ± 53	91 ± 3
	Recurrent SAC+RAD	323 ± 30	279 ± 111	420 ± 27	304 ± 22	182 ± 88	228 ± 34	524 ± 35
	CURL	391 ± 30	102 ± 20	432 ± 15	233 ± 13	648 ± 32	253 ± 40	678 ± 35
	CoRe	**586 ± 30**	**798 ± 30**	**499 ± 22**	**423 ± 22**	**713 ± 81**	340 ± 60	**744 ± 40**
	CoRe (1M steps)	722 ± 28	909 ± 10	590 ± 17	569 ± 19	823 ± 75	552 ± 83	889 ± 26

(b) Benchmark medium. Mean reward ± Standard Error. **Bold** indicates best results at 500K steps.

	Method	Mean	BiC-Catch	C-swingup	C-run	F-spin	R-easy	W-walk
Static	SAC+RAD	113 ± 12	96 ± 14	272 ± 11	21 ± 15	169 ± 92	93 ± 6	25 ± 1
	QT-Opt+RAD	165 ± 15	172 ± 12	297 ± 7	130 ± 7	234 ± 67	94 ± 16	63 ± 3
	QT-Opt+DrQ	170 ± 11	169 ± 25	283 ± 5	124 ± 9	266 ± 51	112 ± 16	64 ± 4
	Recurrent SAC+RAD	211 ± 26	37 ± 21	394 ± 28	244 ± 24	124 ± 107	91 ± 7	378 ± 19
	CURL	300 ± 28	124 ± 33	304 ± 20	277 ± 12	621 ± 21	74 ± 22	402 ± 78
	CoRe	**561 ± 29**	**762 ± 27**	**509 ± 14**	**402 ± 15**	**880 ± 9**	**219 ± 25**	**593 ± 26**
	CoRe (1M steps)	690 ± 26	743 ± 88	634 ± 11	526 ± 17	924 ± 5	543 ± 56	766 ± 30
Dynamic	SAC+RAD	89 ± 5	139 ± 7	192 ± 6	14 ± 2	63 ± 24	93 ± 6	31 ± 2
	QT-Opt+RAD	103 ± 3	132 ± 20	241 ± 7	52 ± 3	25 ± 6	105 ± 10	64 ± 2
	QT-Opt+DrQ	102 ± 5	114 ± 22	243 ± 5	54 ± 2	26 ± 5	108 ± 5	65 ± 1
	Recurrent SAC+RAD	198 ± 20	257 ± 89	277 ± 19	244 ± 17	90 ± 47	112 ± 17	211 ± 15
	CURL	223 ± 14	136 ± 24	320 ± 11	170 ± 10	163 ± 37	222 ± 37	328 ± 19
	CoRe	**480 ± 23**	**706 ± 39**	**354 ± 26**	**354 ± 10**	**540 ± 73**	**445 ± 48**	**479 ± 31**
	CoRe (1M steps)	684 ± 24	832 ± 22	483 ± 29	490 ± 13	810 ± 60	801 ± 32	686 ± 42

(c) Benchmark hard. Mean reward ± Standard Error. **Bold** indicates best results at 500K steps.

	Method	Mean	BiC-Catch	C-swingup	C-run	F-spin	R-easy	W-walk
Static	Recurrent SAC+RAD	168 ± 17	115 ± 26	369 ± 10	166 ± 8	20 ± 18	80 ± 20	258 ± 30
	CURL	202 ± 16	163 ± 36	199 ± 30	239 ± 12	244 ± 53	121 ± 16	247 ± 53
	CoRe	**499 ± 28**	**710 ± 37**	**447 ± 10**	**339 ± 13**	**809 ± 14**	**197 ± 21**	**490 ± 15**
	CoRe (1M steps)	638 ± 27	875 ± 9	554 ± 11	469 ± 21	898 ± 12	386 ± 39	645 ± 29
Dyn	Recurrent SAC+RAD	157 ± 17	214 ± 60	193 ± 4	143 ± 9	131 ± 78	101 ± 7	162 ± 16
	CURL	95 ± 9	103 ± 18	192 ± 6	78 ± 13	5 ± 2	73 ± 13	119 ± 25
	CoRe	**307 ± 22**	**436 ± 48**	**257 ± 18**	**200 ± 11**	**364 ± 83**	**234 ± 61**	**353 ± 25**
	CoRe (1M steps)	467 ± 29	562 ± 65	350 ± 26	345 ± 15	620 ± 97	419 ± 90	505 ± 17

does not contrastively predict the next observation, or model the dynamics and reward. Comparisons are made at 500K environment steps, though we report our results at 1M environment steps as well to show that our model continues to improve with more steps. Performance is averaged over 5 random seeds, and 20 validation episodes at each checkpoint. On the easy benchmark (Table 5a), CoRe outperforms other methods in all tasks, except reacher. As discussed in Section 3.3, this points to a key limitation of contrastive learning-based methods, which is that they tend to remove constant information (such as the goal location for reacher). However, at 1M steps, the performance on reacher is much better, showing that the model is able to eventually solve the task.

On the medium benchmark (Table 5b) CoRe outperforms other methods across all tasks, showing that it can deal with the presence of more distractions. The performance on the reacher tasks improves slightly over the easy setting, which shows that having more variation in the distractions actually helps training, whereas it hurts the baseline methods. We also report results on the hard setting (Table 5c), which is documented in the DCS codebase. Stone et al. (2021) do not report model-free baselines for this setting, presumably because the baseline models fail to train reasonable policies at all. However, CoRe is able to get off the ground and get reasonable performance even in this setting. Training to 1M steps continues to improve results.

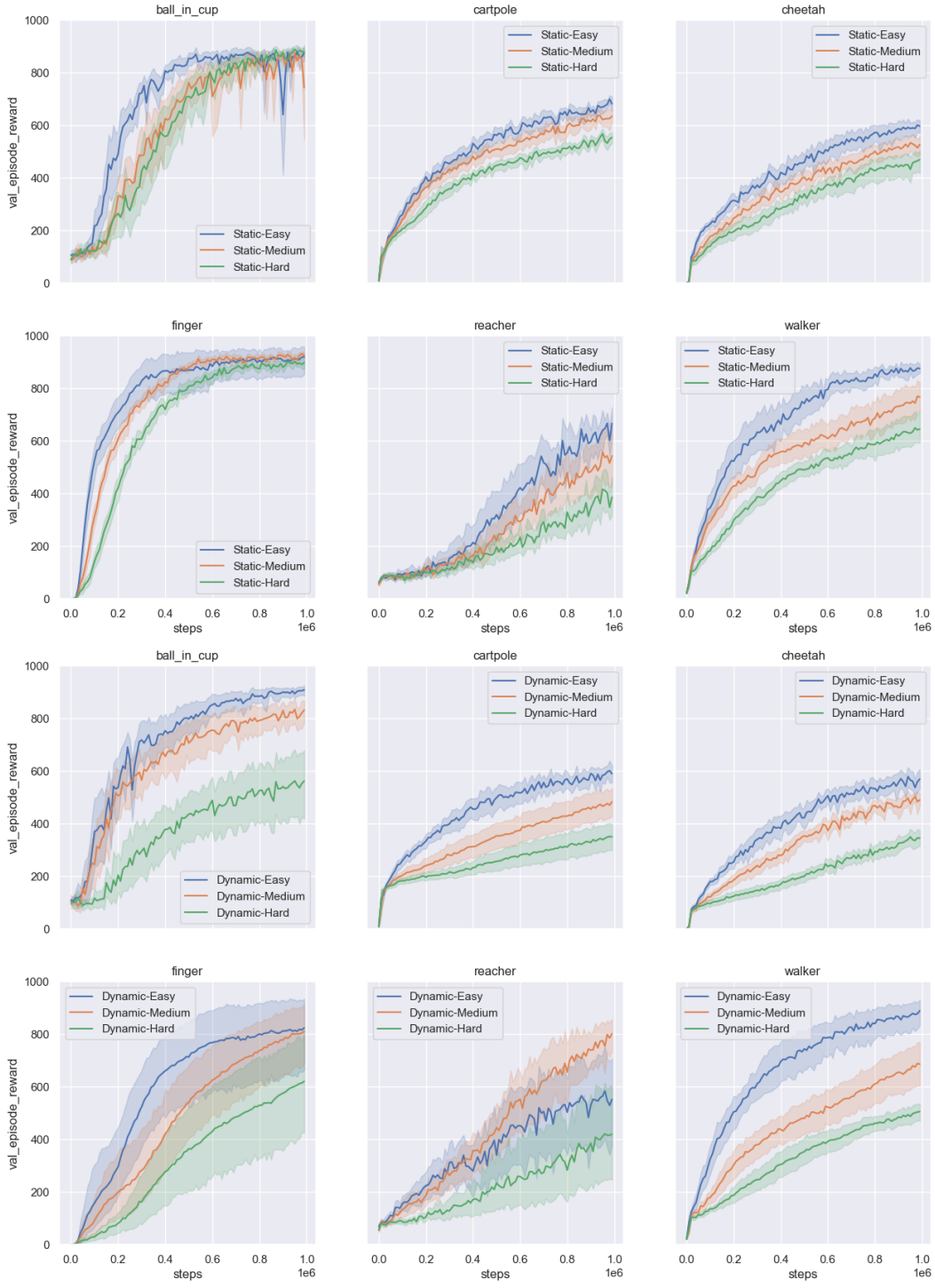
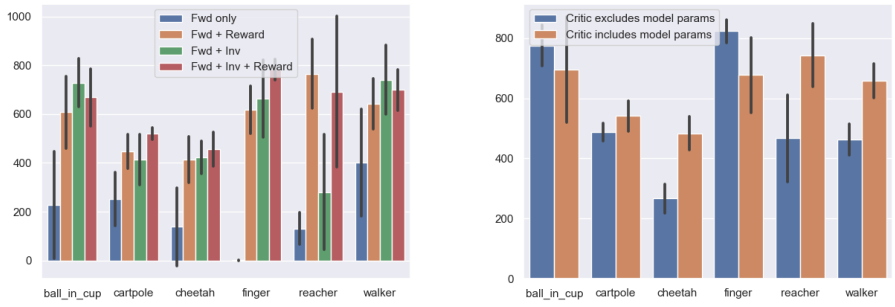


Figure 3: Progression of validation reward with training steps on all distraction settings in the Distracting Control Suite. Top two rows are for the static setting and bottom two for dynamic. Each plot shows easy, medium, and hard difficulty levels.

Figure 3 shows the progression of validation reward for CoRe over 1M environment steps for all tasks and distractions settings. We can see that the hard-dynamic setting (green curves in the bottom two rows) is the hardest to fit because the performance increases slowly. However, in most cases



(a) Inverse dynamics and reward prediction.

(b) Optimizing $J_Q(\Theta, \theta_i)$ vs $J_Q(\theta_i)$

Figure 4: Additional ablation results. **Left** In addition to forward dynamics, inverse dynamics prediction and reward prediction improve performance. **Right:** Optimizing model parameters Θ using the critic loss is beneficial.

the performance for that setting is still improving at 1M steps, and is likely to get better with more training. Our model struggles on the reacher task in terms of performance reported at 500K and even 1M steps, but we can see from these plots that the model is likely to continue improving if trained beyond 1M steps in both static and dynamic settings. In our experiments, we did not tune hyper-parameters specifically for each task, so it is possible that some tasks can benefit from further tuning. In particular, for the reacher task, a bigger batch-size can help since that is the only way to get access to diverse target positions.

C Additional Ablations

C.1 Importance of inverse dynamics and reward prediction

In addition to forward dynamics prediction, the proposed CoRe model includes reward prediction and inverse dynamics prediction as auxiliary tasks. In this section, we present comparisons to ablated versions of our model that remove one or both of these tasks. In Figure 4a we can see that removing both tasks (Fwd only) is significantly worse than having both (Fwd + inv + reward). Having any one of these alone is a big improvement on all tasks except reacher, where adding reward is much more important than inverse dynamics. It is interesting to see that in the absence of reward, adding inverse dynamics prediction (Fwd + inv) improves over having forward dynamics only. Asking the model to predict the action that takes the agent from one state to the next is a different way of expressing the model dynamics, compared to predicting the next state given the current state and action. The fact that asking the model to do both simultaneously gives a boost in performance indicates that inverse dynamics prediction shapes the latent state in ways that are complementary to forward dynamics.

C.2 Importance of updating Θ using critic loss

In our model we optimize the parameters Θ of the world model (observation encoder and RSSM) using the critic loss J_Q. This is similar to the choice made in SLAC (Lee et al., 2020) and DBC (Zhang et al., 2021). However, a reasonable alternative could be to optimize Θ only using J_M and keep the critic training separate. This would amount to separating the world model from the controller. As shown in Figure 4b, when excluding Θ from the critic vs including it, exclusion performs comparably on two tasks (ball_in_cup, cartpole), better on one (finger) and worse on three (cheetah, reacher, and walker). Overall, the inclusion regime works better. Therefore, we chose to include Θ in the critic. In future work, an important direction to explore is the exclusion regime, especially in the multi-task setting, because separating the controller from the world model enables separating general understanding of the world from task-specific control policies, which is key to generalization.

C.3 Robustness to β when reconstructing from prior vs posterior

CoRe predicts the next observation's feature from the *prior* latent state $\hat{z}_t$, making it different from sequential VAE-based models like SLAC (Lee et al., 2020) and Dreamer (Hafner et al., 2020) where

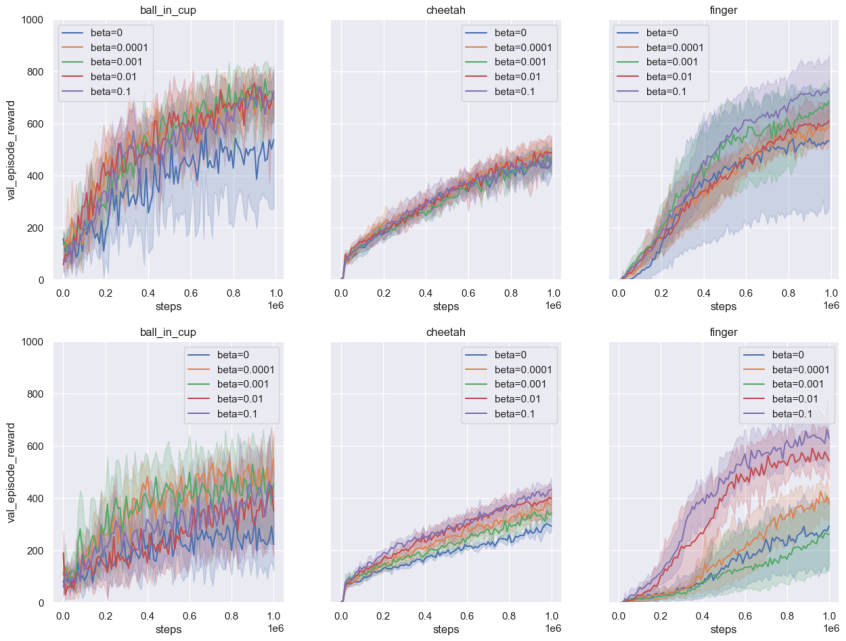


Figure 5: **Top:** Performance of CoRe for different values of the KL-loss weight β. **Bottom** Performance of a variant in which contrastive prediction is done from the posterior latent state, rather than the prior. We can see higher performance variance across values of β when the posterior latent state is used.

the *posterior* latent state is used to reconstruct the observation. We argued in the paper that doing so makes the model more robust to the choice of β, the weight applied to the KL-term during optimization. In this experiment, we verify this argument by comparing CoRe with a variant where the posterior latent state is decoded to contrastively match the true observation's representation. We train both models with five values of β and 5 different seeds each. In Figure 5 we can see that the posterior version (bottom row) has more variance in performance across different values of β. The proposed CoRe model (top row) is more stable, and hence, easier to train.

REFERENCES

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of The 2nd Conference on Robot Learning*, 2018.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *International Conference on Machine Learning*, 2020a.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, 2020b.

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*, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 2019.

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite – a challenging benchmark for reinforcement learning from pixels. *arXiv preprint arXiv:2101.02722*, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033. IEEE, 2012.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. 2019.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.

Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021.