A Parameters and implementation

We use as single set of parameters throughout all experimental evaluations. The general model architecture follows Hafner et al. [7], where the variational autoencoder from Ha and Schmidhuber [4] is combined with the RSSM from Hafner et al. [29]. We extend their default parameters by the ensemble size $M = 5$, the initial UCB trade-off parameter $\beta_{ini} = 0.0$, and the per-episode linear UCB growth rate $\delta = 0.001$. The learning rates for the model, the value function and the policy are 6×10^{-4}, 8×10^{-5}, 2×10^{-4}, respectively, and updates are computed with the Adam optimizer [55]. Throughout all experiments, the online phase consists of 1000 environment interactions with an action repeat of 2, while the offline phase consists of 200 learning updates.

An overview of the range of hyper-parameter values that were investigated is provided in Table 2. Not all possible pairings were considered and suitable combinations were determined by inspection, while the best pairing was selected empirically. Our implementation builds on Dreamer (`https://github.com/danijar/dreamer`) and the remaining parameters are set to their default values. Experiments were conducted on 4 CPU cores in combination with 1 GPU (NVIDIA V100). We will make the underlying codebase publicly available.

Param.	Values
LR_π	$\left[8 \times 10^{-5},\ 2 \times 10^{-4}\right]$
Steps	$[100,\ 200]$
β_{ini}	$[-0.1,\ 0.0,\ 0.1,\ 0.3,\ 0.5]$
δ	$\left[+\{-10^{-3}, 0.0, 10^{-3}, 2 \cdot 10^{-3}\}, \times\{1.01, 1.015\}\right]$

Table 2: Hyper-parameters considered during training.

B Network Architectures

The base network architectures employed throughout this paper are provided in Table 3. Each particle is assigned a distinct instance of its associated models. In the following, we briefly comment on how the two parts of the transition model interact and provide further insights into the remaining models.

Transition model The transition model follows the recurrent state space model (RSSM) architecture presented in Hafner et al. [7, 29]. The RSSM propagates model states consisting of a deterministic and a stochastic component, respectively denoted by $s_{t,d}$ and $s_{t,s}$ at time t. The stochastic component $s_{t,s}$ is represented as a diagonal Gaussian distribution. The transition model then leverages the *imagine 1-step* method to predict priors for the associated mean and standard deviation, $(\mu_{t,s}^{prior}, \sigma_{t,s}^{prior})$, based on the previous model state and applied action. In the presence of observations, the *observe 1-step* method can be leveraged to convert prior estimates into posterior estimates, $(\mu_{t,s}^{post}, \sigma_{t,s}^{post})$. The transition model may then propagate posteriors based on a context sequence using both the *imagine 1-step* and *observe 1-step* methods, from which interactions can be imagined by propagating prior estimates based on the *imagine 1-step* method. Each particle uses a transition model that follows the presented network architecture, but possesses distinct parameters.

Encoder model The encoder parameterization follows the architectural choices presented in Ha and Schmidhuber [4]. The encoder generates embeddings based on 64×64 RGB image observations.

Observation model The observation model follows the decoder architecture presented in Ha and Schmidhuber [4]. The image observations are reconstructed from the associated model states s_τ.

Reward and value model Rewards and values are both predicted as scalar values from fully-connected networks that operate on the associated model states s_τ, similar to Hafner et al. [7]. Each particle uses a pairing of a reward model and a value model with distinct sets of parameters.

Action model The action model follows Hafner et al. [7], where the mean μ_a is rescaled and passed through a tanh to allow action saturation. It is combined with a softplus standard deviation based on σ_a and the resulting Normal distribution is squashed via a tanh (see Haarnoja et al. [18]).

Layer Type	Input (dimensions)	Output (dimensions)	Additional Parameters
Transition model (*imagine 1-step*)			
Dense	$s_{\tau-1,s}$ (30), $a_{\tau-1}$ (n_a)	$\text{fc}^1_{t,i}$ (200)	a=ELU
GRU	$\text{fc}^1_{t,i}$ (200), $s_{\tau-1,d}$ (200)	rs_τ (200), $s_{\tau,d}$ (200)	a=tanh
Dense	rs_τ (200)	$\text{fc}^2_{t,i}$ (200)	a=ELU
Dense	$\text{fc}^2_{t,i}$ (200)	$\mu^{prior}_{\tau,s}$ (30), $\sigma^{prior}_{\tau,s}$ (30)	a=None
Transition model (*observe 1-step*)			
Dense	$s_{\tau,d}$ (200), z_τ (1024)	$\text{fc}^1_{t,o}$ (200)	a=ELU
Dense	$\text{fc}^1_{t,o}$ (200)	$\mu^{post}_{\tau,s}$ (30), $\sigma^{post}_{\tau,s}$ (30)	a=None
Encoder model			
Conv2D	obs (64, 64, 3)	cv1 (31, 31, 32)	a=ReLU, s=2, k=(4,4)
Conv2D	cv1 (31, 31, 32)	cv2 (14, 14, 64)	a=ReLU, s=2, k=(4,4)
Conv2D	cv2 (14, 14, 64)	cv3 (6, 6, 128)	a=ReLU, s=2, k=(4,4)
Conv2D	cv3 (6, 6, 128)	cv4 (2, 2, 256)	a=ReLU, s=2, k=(4,4)
Reshape	cv4 (2, 2, 256)	z_τ (1, 1, 1024)	
Observation model			
Dense	$s_{\tau,d}$ (200), $s_{\tau,s}$ (30)	fc^1_o (1, 1, 1024)	a=None
Deconv2D	fc^1_o (1, 1, 1024)	dc1 (5, 5, 128)	a=ReLU, s=2, k=(5,5)
Deconv2D	dc1 (5, 5, 128)	dc2 (13, 13, 64)	a=ReLU, s=2, k=(5,5)
Deconv2D	dc2 (13, 13, 64)	dc3 (30, 30, 32)	a=ReLU, s=2, k=(6,6)
Deconv2D	dc3 (30, 30, 32)	dc4 (64, 64, 3)	a=ReLU, s=2, k=(6,6)
Reward model			
Dense	$s_{\tau,d}$ (200), $s_{\tau,s}$ (30)	fc^1_r (400)	a=ELU
Dense $\times$ 1	$\text{fc}^{\{1\}}_r$ (400)	$\text{fc}^{\{2\}}_r$ (400)	a=ELU
Dense	fc^2_r (400)	fc^3_r (1)	a=ELU
Value model			
Dense	$s_{\tau,d}$ (200), $s_{\tau,s}$ (30)	fc^1_v (400)	a=ELU
Dense $\times$ 2	$\text{fc}^{\{1,2\}}_v$ (400)	$\text{fc}^{\{2,3\}}_v$ (400)	a=ELU
Dense	fc^3_v (400)	fc^4_v (1)	a=ELU
Action model			
Dense	$s_{\tau,d}$ (200), $s_{\tau,s}$ (30)	fc^1_a (400)	a=ELU
Dense $\times$ 3	$\text{fc}^{\{1,2,3\}}_a$ (400)	$\text{fc}^{\{2,3,4\}}_a$ (400)	a=ELU
Dense	fc^4_a (400)	μ_a (n_a), σ_a (n_a)	a=ELU

Table 3: General network architectures of the underlying models. We note that repeated layers have been condensed with Dense $\times$ i referring to application of the same dense layer architecture i times. Parameter abbreviations: a=activation, k=kernel, and s=stride. Adapted from Hafner et al. [7].

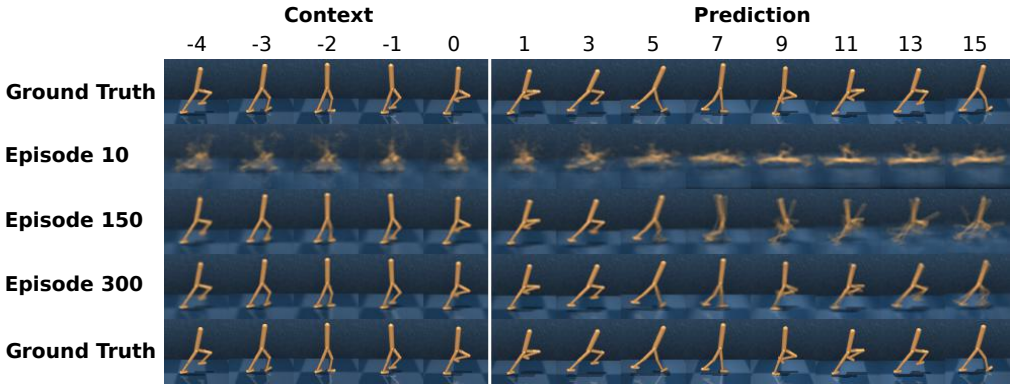

Figure 4: Motion pattern of the Walker with low predictive uncertainty. The agent is provided with 5 contextual images and predicts forward for 15 steps (preview horizon), at different stages of training. The regular walking pattern is well-explored and only induces little deviation in the ensemble. This motion is desirable and the agent should focus on reducing its uncertainty over environment behavior.

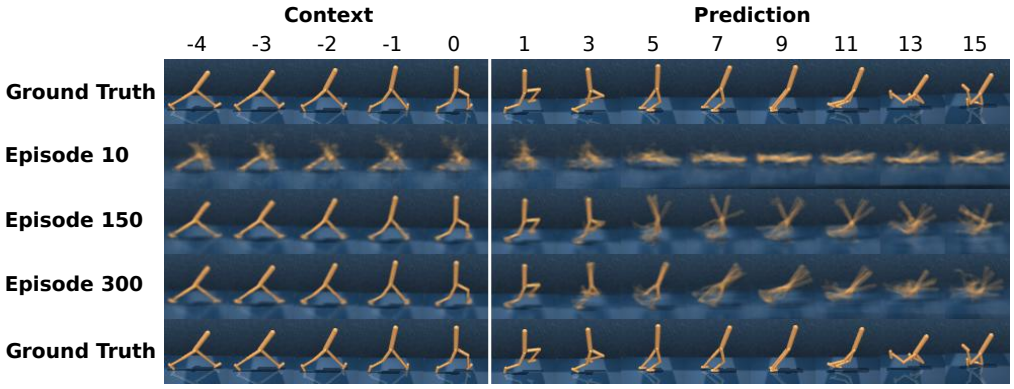

Figure 5: Motion pattern of the Walker with high predictive uncertainty. The agent is provided with 5 contextual images and predicts forward for 15 steps (preview horizon), at different stages of training. The irregular falling pattern has not been extensively explored and high uncertainty remains in the ensemble. This motion is undesirable and the agent should not focus on reducing its uncertainty.

C Prediction uncertainty

We provide an illustrative visualization of how the prediction uncertainty in the ensemble evolves during model training. The ensemble is provided with context from a sequence of 5 consecutive images and then predicts forward in an open loop fashion for 15 steps (preview horizon). The ground truth sequence is compared to ensemble predictions after 10, 150, and 300 episodes of agent training.

Figures 4 and 5 show two different motion patterns for the Walker Walk task. The motion in Figure 4 can be described as a regular walking pattern. At the beginning of model training, the agent will have mostly observed itself falling to the ground and, in combination with a poorly trained policy, the ensemble predictions place the agent on the ground in a variety of configurations. After 150 episodes, short-term uncertainty has been significantly reduced, while considerable uncertainty remains at the end of the preview window. After 300 episodes, the ensemble predictions align with the ground truth sequence. The agent therefore focused on reducing uncertainty over this desirable motion pattern. This can be contrasted with the results of Figure 5, where uncertainty over an irregular falling pattern remains even after 300 episodes. The falling motion is undesirable, and while the ensemble predictions agree on a fall being imminent, no significant amount of effort was spent on identifying exactly how the agent would fall. We can observe similar results on the Cheetah Run task for a running motion pattern in Figure 6 and a falling motion pattern in Figure 7. However, the lower complexity Cheetah dynamics seem to allow for more precise predictions than on the Walker task.

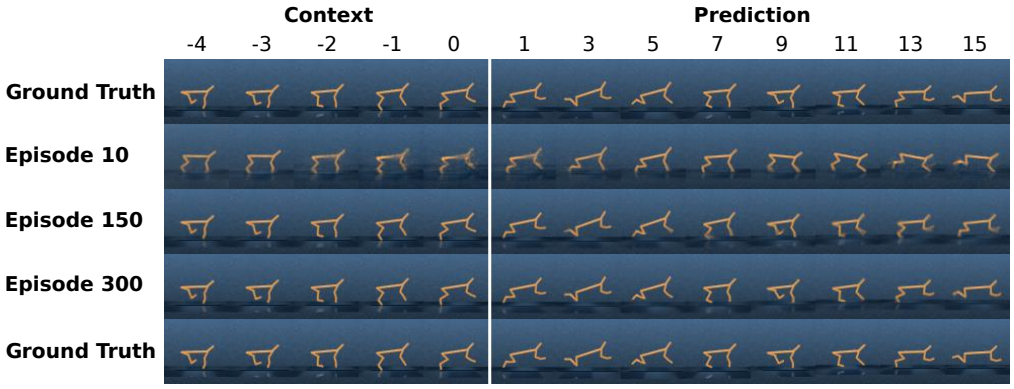

Figure 6: Motion pattern of the Cheetah with low predictive uncertainty. The agent is provided with 5 contextual images and predicts forward for 15 steps (preview horizon), at different stages of training. The regular running pattern is well-explored and only induces little deviation in the ensemble. This motion is desirable and the agent should focus on reducing its uncertainty over environment behavior.

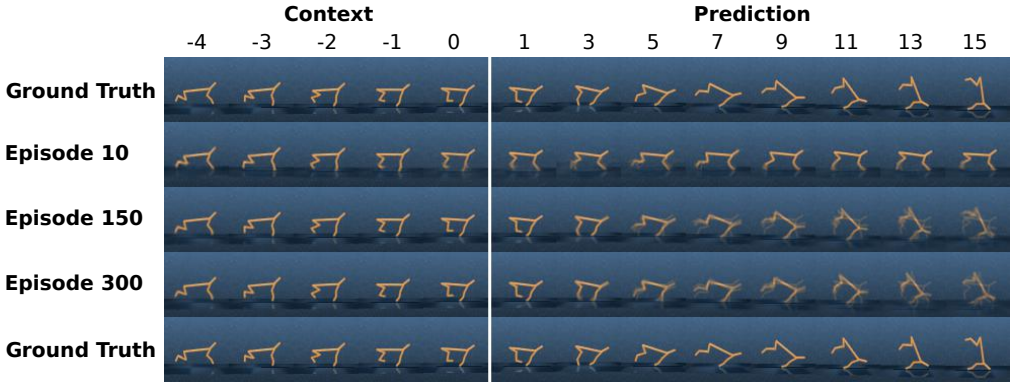

Figure 7: Motion pattern of the Cheetah with high predictive uncertainty. The agent is provided with 5 contextual images and predicts forward for 15 steps (preview horizon), at different stages of training. The irregular falling pattern has not been extensively explored and uncertainty remains in the ensemble. This motion is undesirable and the agent should not focus on reducing its uncertainty.

D Baselines

The baseline performance data for DrQ was taken from Kostrikov et al. [15], the ones for D4PG and A3C from Tassa et al. [13], while the data for Dreamer was generated by running the official TensorFlow 2 implementation of Hafner et al. [7]. It should be noted that both DrQ and D4PG use 84×84 image observations, whereas LOVE and Dreamer use 64×64 image observations. Larger resolution provides more fine-grained information, which potentially translates to improved planning. Furthermore, DrQ continuously refines its policy online, while the other algorithms only do so offline.

E Bugtrap extended

We provide additional occupancy maps for the bug trap environment in Figure 8. The environment provides no reward feedback and assesses the agent's ability to actively search for informative feedback through intrinsic motivation. Furthermore, the environment geometry makes exploration of the outside area difficult. In the absence of useful mean performance estimates, LOVE leverages uncertainty-guided exploration to query interactions. This allows for escaping in 5 out of 6 trials and achieving the largest area coverage (column 2). LVE does not leverage uncertainty estimates and only escapes during 3 trials (column 3), while displaying a highly reduced area coverage (rows 1 and 3). Similarly, random exploration allows the Dreamer agent to only escape in 2 instances (column 4).

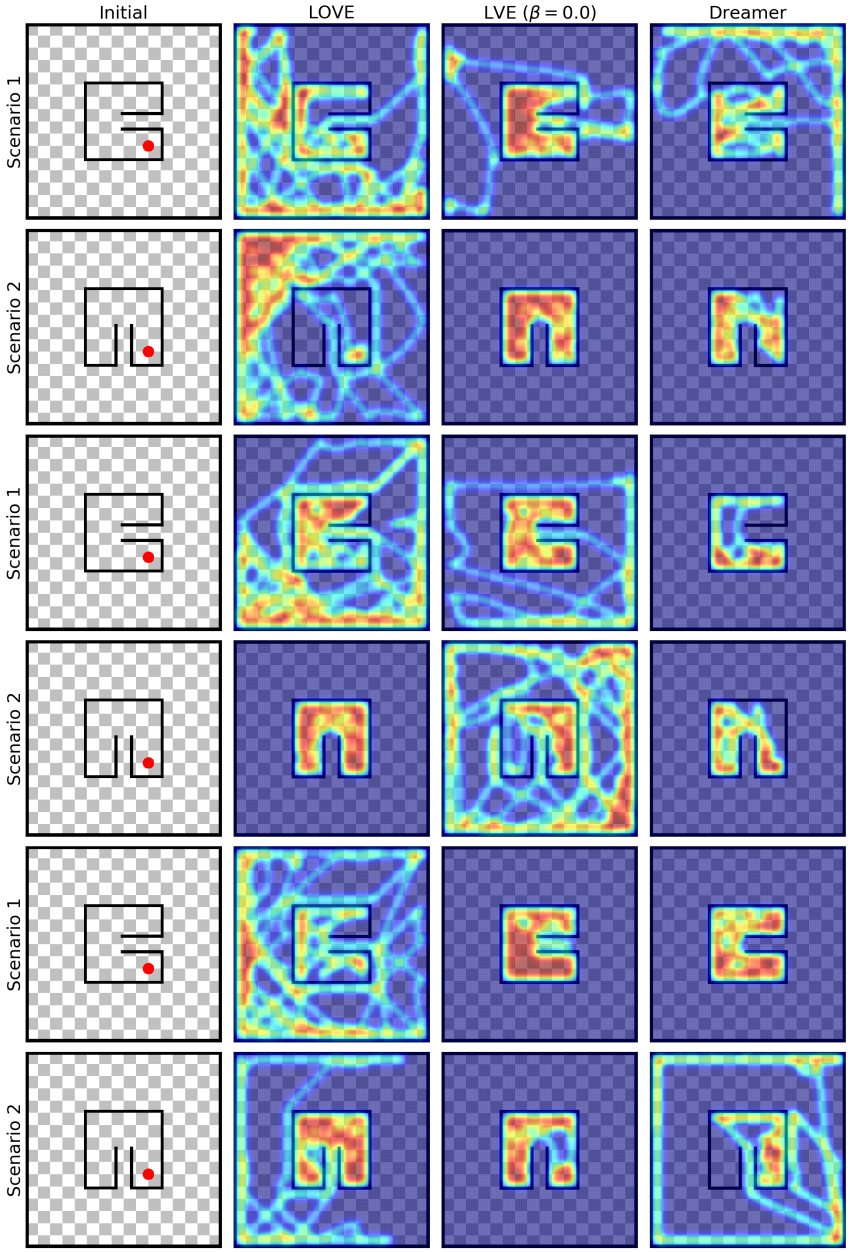

Figure 8: Occupancy maps of the bug trap environment for two scenarios and three random seeds. In the absence of reward feedback, the uncertainty-guided exploration allows LOVE to escape during 5 out of 6 runs while achieving the highest area coverage in search of non-zero reward feedback. LVE removes optimistic exploration and as a result only escapes during 3 runs, while significantly reducing area coverage. A similar pattern can be observed for the randomly exploring Dreamer agent.

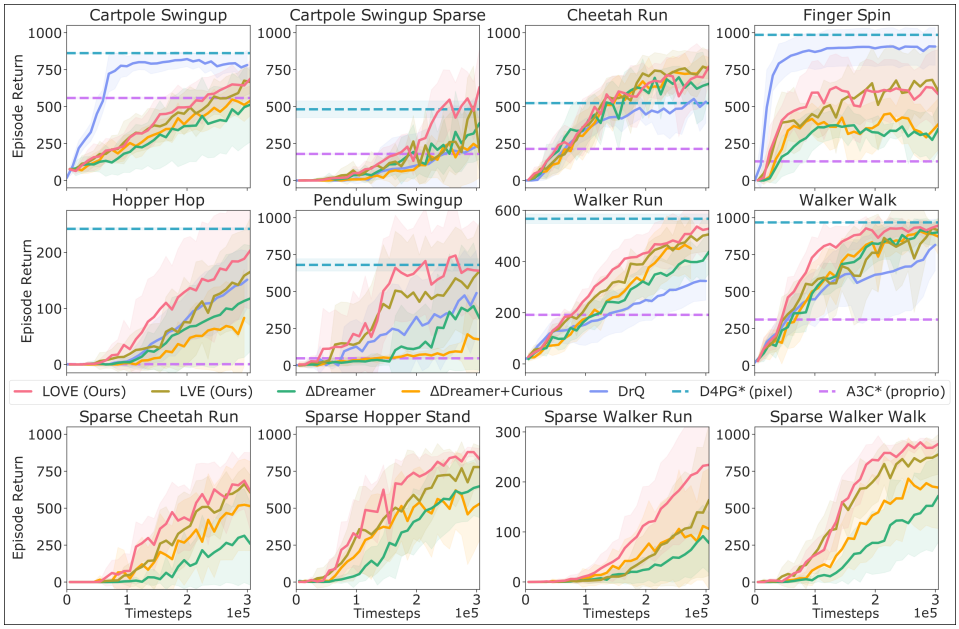

Figure 9: DeepMind Control Suite. We evaluate performance over 300 episodes on 9 seeds. Solid lines indicate mean performance and shaded areas indicate one standard deviation. LOVE performs competitively and improves sample efficiency particularly under sparse reward feedback. Temporally-extended optimism helps LOVE in actively exploring uncertain returns, providing an advantage over LVE. Formulating intrinsic motivation in reward-space enables LOVE to identify uncertain interactions conducive to solving the task, providing an advantage over the curiosity baseline ΔDreamer+Curious. *D4PG, A3C: converged results at 10^8 environment steps as reference.

F Benchmarking on DeepMind Control Suite

Figure 9 provides results for benchmarking performance over 300 episodes. Performance is evaluated on 9 seeds, where solid lines indicate the mean and shaded areas correspond to one standard deviation. LOVE's ability to explore uncertain long-term returns is particularly well-suited to reward structures that include sparsity. In particular, LOVE outperforms the curiosity baseline ΔDreamer+Curious as LOVE does not get distracted by uncertain but irrelevant unexpected environment behavior. Overall, LOVE yields the best performance across all tasks.

G Ablation study: Dreamer

We compare performance to ΔDreamer, a variation that uses our changes to the default parameters. Figure 10 indicates that performance improves on several tasks, while deteriorating on Finger Spin. LOVE outperforms ΔDreamer on the majority of tasks. It can thus be concluded that increased information propagation generally affects performance favourably. However, relying on a single model can propagate simulation bias into the policy and in turn impede efficient learning. This could serve as an explanation for the unchanged performance on the not fully observable Cartpole Swingup tasks, as well as the deteriorating performance on the high-frequency Finger Spin task.

H Ablation study: planning horizon

We investigate increasing the planning horizon used during latent imagination. Longer horizons shift performance estimation from values towards rewards, which can be advantageous when the value function has not been sufficiently learned. Prediction quality over long horizons relies on accurate dynamics rollouts. Figure 11 indicates that an intermediate horizon is a good trade-off. We note that the two Walker tasks had not completed at the time of submission, but provide a visible trend.

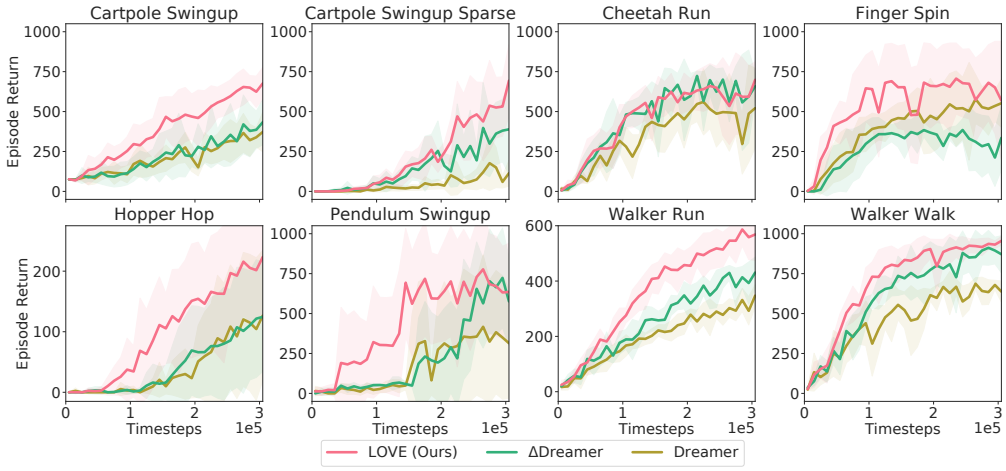

Figure 10: Comparison to Dreamer with adapted policy learning rate and training steps (ΔDreamer). The changes improve performance of Dreamer on some environments, while significantly decreasing performance on the Finger Spin task. LOVE still outperforms ΔDreamer on the majority of tasks.

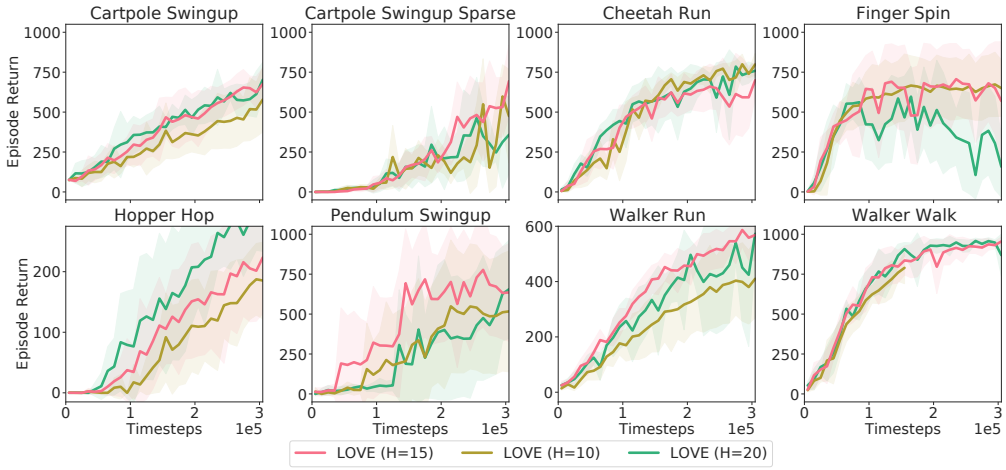

Figure 11: LOVE under variation of the planning horizon.

I Ablation study: β schedule

We investigate variations of the beta schedule, initializing either with a negative (initially pessimistic) or a positive value (initially optimistic). The former variation penalizes uncertainty in the beginning and then transitions to become optimistic, while the latter seeks out uncertainty from the start. Based on Figure 12, we notice that terminal performance is mostly similar. The initially pessimistic agent exhibits reduced performance on the sparse pendulum, where it only explores well after it transitions to optimism (Episode 100), and improved performance on the challenging Hopper task, where initial pessimism potentially guards against local optima. Our choice of parameters tries to mitigate unfounded optimism during initialization (initial value 0), while encouraging exploration throughout the course of training (linear increase). Particularly at initialization, a strong positive UCB parameter may amplify random parameter noise as the agent will not have recovered meaningful representations or sufficiently propagated learned values, yet. Here, we chose the same values for all tasks, but one could imagine task-specific choices (negative beta for safe-RL, positive beta for optimistic exploration).

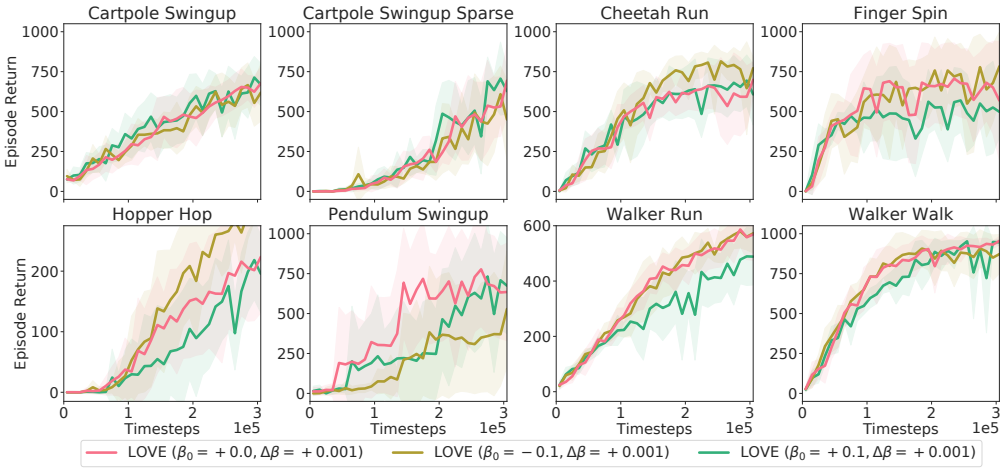

Figure 12: LOVE under variation of the beta schedule.

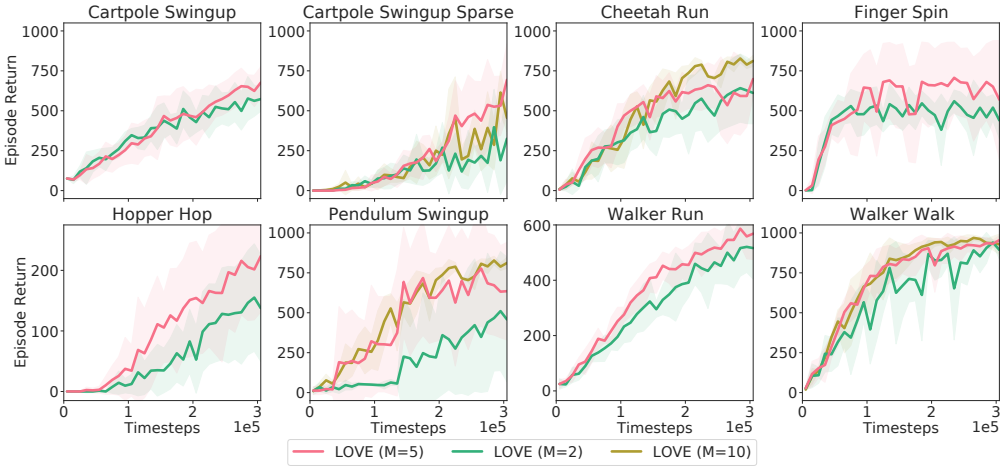

Figure 13: LOVE under variation of the ensemble size.

J Ablation study: ensemble size

We investigate variation of the ensemble size. A smaller ensemble generates uncertainty estimates that are more susceptible to bias in the ensemble members and may even generate misleading estimates. Figure 14 demonstrates that a smaller ensemble (M=2) impacts performance unfavorably. We also provide data for a larger ensemble (M=10) on Cartpole Sparse, Cheetah, Pendulum and Walker Walk due to computation constraints. Generally, increasing the number of ensemble members increases the computational burden and we find the common literature choice of M=5 to perform sufficiently well.

K λ-returns

The value functions are trained with λ-return targets, which are computed according to

$$\begin{aligned} V_\lambda(s_\tau) &\doteq (1-\lambda)\sum_{n=1}^{H-1}\lambda^{n-1}V_N^n(s_\tau) + \lambda^{H-1}V_N^H(s_\tau), \\ V_N^k(s_\tau) &\doteq E_{q_\theta,q_\phi}\left(\sum_{n=\tau}^{h-1}\gamma^{n-\tau}r_n + \gamma^{h-\tau}v_\psi(s_h)\right), \end{aligned} \tag{8}$$

with $h = \min(\tau+k, t+H)$.

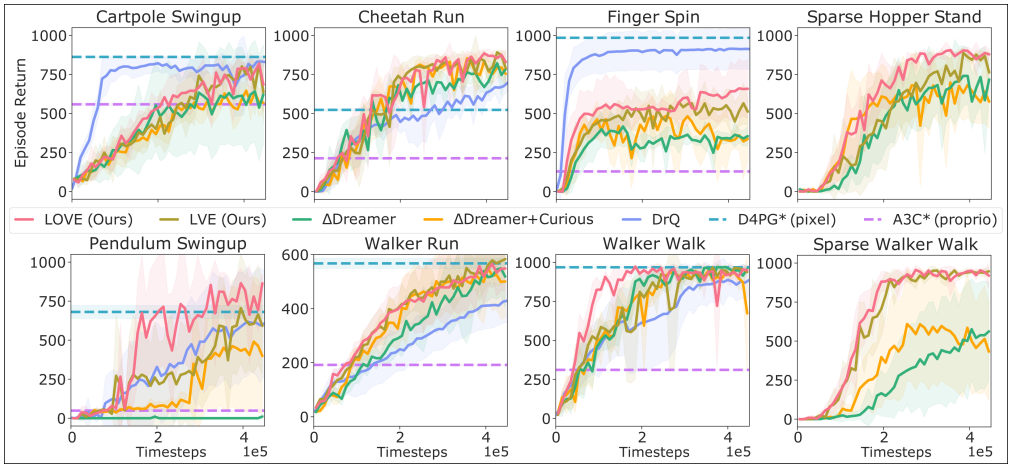

Figure 14: Evaluation of asymptotic performance based 450k environment steps. We observe that LOVE approaches or exceeds D4PG performance at 10^8 environment steps on most environments.

L Asymptotic performance

We provide episode returns over 450k environment steps to better assess asymptotic performance. We consider a subset of the environments across 4 seeds. We observe that LOVE converges to or beyond converged D4PG performance with the exception on Finger Spin. LOVE further significantly outperform the Dreamer baselines. Generally, the comparatively low performance of the model-based agents on Finger Spin could indicate that the required high frequency behavior can be difficult to learn based on an explicit model. Throughout, we employed the same UCB trade-off parameter schedule as described in Section A underlining that the agent does not get distracted by tangential uncertainty.

M Interleaved Exploitation

We briefly evaluate explicitly interleaving the exploration and exploitation policy during learning, sampling from either policy with a probability of $p(\pi_{\phi_\cdot}) = 0.5$ with $\phi_\cdot \in \{\phi_{aq}, \phi_{ev}\}$. Due to computation constraints we limit ourselves to 4 seeds on the Cheetah Run and Walker Walk tasks. From Figure 15, we observe that LOVE's implicit exploration-exploitation trade-off (red) and interleaving explicit exploitation (green) perform similarly, with a slight edge for LOVE's implicit trade-off on the Walker task and for explicit exploitation on the Cheetah task. Generally, the effect of interleaving explicit exploitation to generate samples may depend on the nature of the task and the reward density.

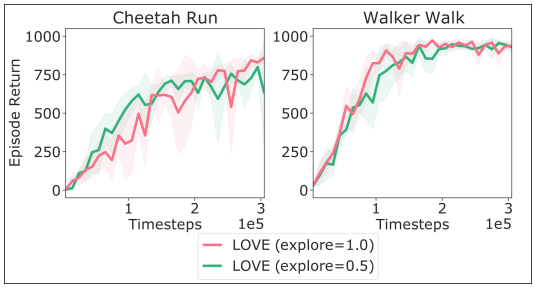

Figure 15: Interleaving action samples from the acquisition policy and the evaluation policy for generating environment interactions.

N Network Initialization

We briefly evaluate changes to the kernel and bias initializers of the reward and value networks. The default initializers are Glorot-Uniform (GU) for the kernels and Zero (ZR) for the biases. For either, we consider the Variance-Scaling initializer (VS) as an alternative, where we use the scaling factor $\sigma = 0.333$. We consider 4 seeds each on the Walker Walk task. Based on Figure 16, we observe that changes to either the kernel or bias initializers slightly lower convergence speed while still reaching comparable asymptotic performance. More sophisticated initialization schemes may include explicit regularization of the network parameters towards random anchors as in [56] and could harbor potential for further improving performance.

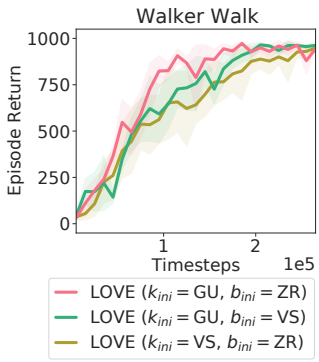

Figure 16: Variation of the kernel and bias initialization.

O Maze exploration

We provide a qualitative comparison to the Explore, Discover and Learn (EDL) algorithm [57] on the reward-free maze task. EDL differs from LOVE in its input-output modalities as EDL considers position control based on state input, whereas LOVE considers acceleration control based on image input. Modifying the EDL agent would be non-trivial and in order to provide fair qualitative insights we integrate our maze domain into their framework with position control from state observations. Particularly, we consider EDL with State Marginal Matching and Sibling Rivalry with the default hyperparameters. Based on Figure 17 (right), we see that the skills discovered by EDL mostly align with the occupancy trances of the LOVE agent, while both achieve greater coverage than LVE.

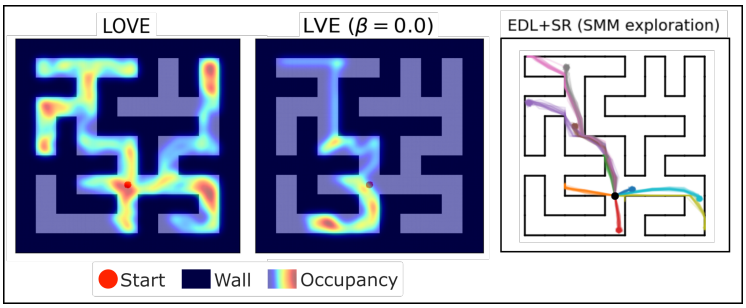

Figure 17: Maze exploration in comparison to the EDL [57] agent.