A. Proof of prop. 2

Proof. We first note that a solution exists to the projection operation, and it is unique, which comes from the strict convexity of the objective (Rao, 1984). The Lagrangian of the temperature-scaled LML projection in eq. (4) is

$$L(y,\nu) = -x^\top y - \tau H_b(y) + \nu(k - 1^\top y). \tag{9}$$

Differentiating eq. (9) gives

$$\nabla_y L(y,\nu) = -x + \tau \log \frac{y}{1-y} - \nu \tag{10}$$

and the first-order optimality condition $\nabla_y L(y^\star, \nu^\star) = 0$ gives $y_i^\star = \sigma(\tau^{-1}(x_i + \nu^*))$, where σ is the sigmoid function. Using lem. 1 as $\tau \to 0^+$ gives

$$y_i^\star = \begin{cases} 1 & \text{if } x_i > -\nu^* \\ 0 & \text{if } x_i < -\nu^* \\ 1/2 & \text{otherwise.} \end{cases} \tag{11}$$

Substituting this back into the constraint $1^\top y^\star = k$ gives that $\pi(x)_k < -\nu^* < \pi(x)_{k+1}$, where $\pi(x)$ sorts $x \in \mathbb{R}^n$ in ascending order so that $\pi(x)_1 \leq \pi(x)_2 \leq \ldots \leq \pi(x)_n$. Thus we have that $y_i^\star = \mathbb{1}\{x_i \geq \pi(x)_k\}$, which is 1 when x_i is in the top-k components of x and 0 otherwise, and therefore the temperature-scaled LML layer approaches the hard top-k function as $\tau \to 0^+$. □

Lemma 1.

$$\lim_{\tau \to 0^+} \sigma(x/\tau) = \begin{cases} 1 & \textit{if } x > 0 \\ 0 & \textit{if } x < 0 \\ 1/2 & \textit{otherwise,} \end{cases} \tag{12}$$

where $\sigma(x/\tau) = (1 + \exp\{-x/\tau\})^{-1}$ *is the temperature-scaled sigmoid.*

B. More details: Simple regression task

Figure 6 (left) shows the convergence of unrolled GD and DCEM on the training data, showing that they are able to obtain the same training loss despite inducing very different energy surfaces. Figure 6 (right) and fig. 7 shows the impact of training gradient descent and DCEM to take 10 inner optimization steps and then ablating the number of inner steps at test-time.

C. More details: Cartpole experiment

In this section we discuss some of the ablations we considered when learning the latent action space for the cartpole task. In all settings we use DCEM to unroll 10 inner iterations that samples 100 candidate points in each iteration and has an elite set of 10 candidates.

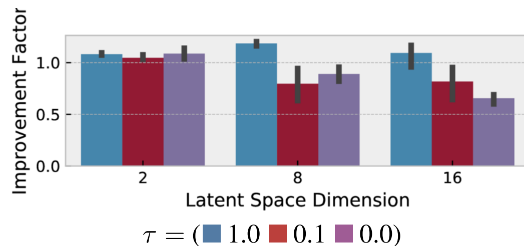

Figure 5. Improvement factor on the ground-truth cartpole task from embedding the action space with DCEM compared to running CEM on the full action space, showing that DCEM is able to recover the full performance. We use the DCEM model that achieves the best validation loss. The error lines show the 95% confidence interval around three trials.

For training, we sample initial starting states of the cartpole and for validation we use a fixed set of initial states. Figure 9 shows the convergence of models as we vary the latent space dimension and temperature parameter, and fig. 5 shows that DCEM is able to fully recover the expert performance on the cartpole. Because we are operating in the ground-truth dynamics setting we measure the performance by comparing the controller costs. We use $\tau = 0$ to indicate the case where we optimize over the latent space with vanilla CEM and then update the decoder with $\nabla_z C(f_\theta^{\text{dec}}(\hat{z}))$, where the gradient doesn't go back into the optimization process that produced $\hat{z}$. This is non-convex min differentiation and is reasonable when $\hat{z}$ is near-optimal, but otherwise is susceptible to making the decoder difficult to search over.

These results show a few interesting points that come up in this setting, which may be different in other settings. Firstly that with a two-dimensional latent space, all of the temperature values are able to find a reasonable latent space at some point during training. However after more updates, the lower-temperature experiments start updating the decoder in ways that make it more difficult to search over and start achieving worse performance than the $\tau = 1$ case. For higher-dimensional latent spaces, the DCEM machinery is necessary to keep the decoder searchable. We notice that just a 16-dimensional latent space for this task can be difficult for learning, one reason this could be is from DCEM having too many degrees of freedom in ways it can update the decoder to improve the performance of the optimizer.

D. More details: Cheetah and walker experiments

For the `cheetah.run` and `walker.walk` DeepMind control suite experiments we start with a modified PlaNet

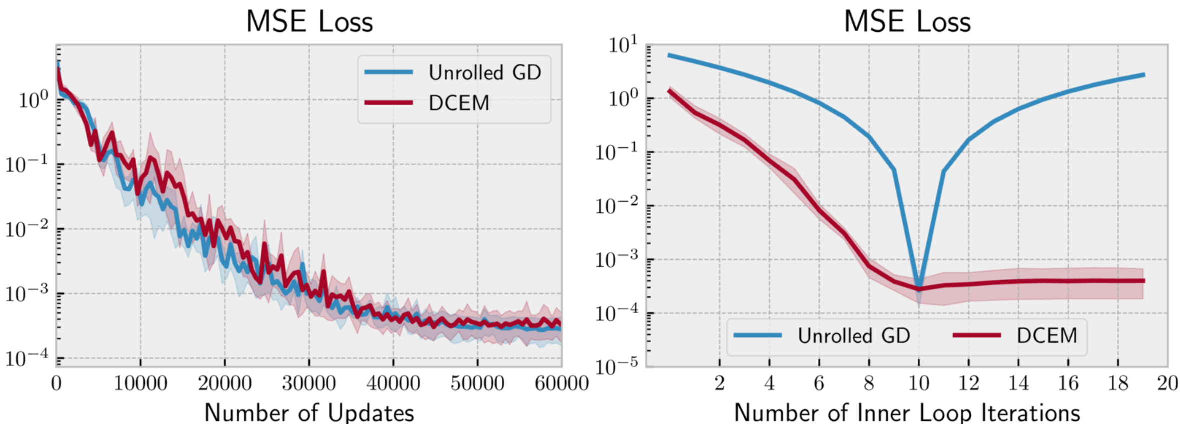

Figure 6. **Left:** Convergence of DCEM and unrolled GD on the regression task. **Right:** Ablation showing what happens when DCEM and unrolled GD are trained for 10 inner steps and then a different number of steps is used at test-time. We trained three seeds for each model and the shaded regions show the 95% confidence interval.

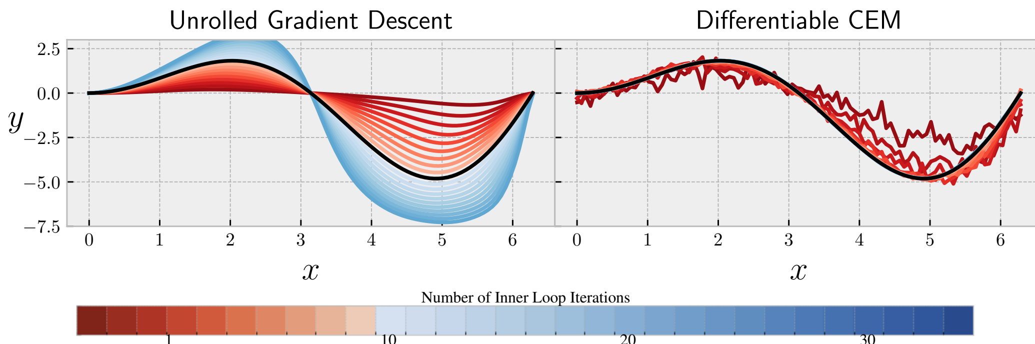

Figure 7. Visualization of the predictions made by ablating the number of inner loop iterations for unrolled GD and DCEM. The ground-truth regression target is shown in black.

(Hafner et al., 2018) architecture without a pixel decoder. We started with this over PETS (Chua et al., 2018) to show that this RSSM is reasonable for proprioceptive-based control and not just pixel-based control. This model is graphically shown in fig. 8 and has 1) a deterministic state model $h_t = f(h_{t-1}, x_{t-1}, u_{t-1})$, 2) a stochastic state model $x_t \sim p(x_t, h_t)$, and 3) a reward model: $r_t \sim p(r_t|h_t, x_t)$. In the proprioceptive setting, we posit that the deterministic state model is useful for multi-step training even in fully observable environments as it allows the model to "push forward" information about what is potentially going to happen in the future.

For the modeling components, we follow the recommendations in Hafner et al. (2018) and use a GRU (Cho et al., 2014) with 200 units as the deterministic path in the dynamics model and implement all other functions as two fully-connected layers, also with 200 units with ReLU activations. Distributions over the state space are isotropic Gaussians with predicted mean and standard deviation. We train the model to optimize the variational bound on the multi-step likelihood as presented in (Hafner et al., 2018) on batches of size 50 with trajectory sequences of length 50. We start with 5 seed episodes with random actions and in contrast to Hafner et al. (2018), we have found that interleaving the model updates with the environment steps instead of separating the updates slightly improves the performance, even in the pixel-based case, which we do not report results on here.

For the optimizers we either use CEM over the full control space or DCEM over the latent control space and use a horizon length of 12 and 10 iterations here. For full CEM, we sample 1000 candidates in each iteration with 100 elite candidates. For DCEM we use 100 candidates in each iteration with 10 elite candidates.

Our training procedure has the following three phases, which we set up to isolate the DCEM additions. We evaluate the models output from these training runs on 100 random episodes in fig. 4 in the main paper. Now that these ideas have been validated, promising directions of future work include trying to combine them all into a single training run

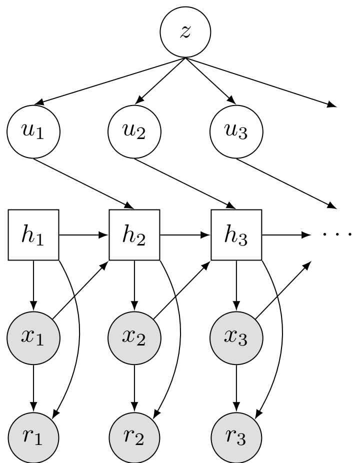

Figure 8. Our RSSM with action sequence embeddings

and trying to reduce the sample complexity and number of timesteps needed to obtain the final model.

Phase 1: Model initialization. We start in both environments by launching a single training run of alg. 3 to get initial system dynamics. These models take slightly longer to converge than in (Hafner et al., 2018), likely due to how often we update our models. We note that at this point, it would be ideal to use the policy loss to help fine-tune the components so that policy induced by CEM on top of the models can be guided, but this is not feasible to do by backpropagating through all of the CEM samples due to memory, so we instead next move on to initializing a differentiable controller that is feasible to backprop through.

Phase 2: Embedded DCEM initialization. Our goal in this phase is to obtain a differentiable controller that is feasible to backprop through.

Our first failed attempt to achieve this was to use offline training on the replay buffer, which would have been ideal as it would require no additional transitions to be collected from the environment. We tried using alg. 2, the same procedure we used in the ground-truth cartpole setting, to generate an embedded DCEM controller that achieves the same control cost on the replay buffer as the full CEM controller. However we found that when deploying this controller on the system, it quickly stepped off of the data manifold and failed to control it — this seemed to be from the controller finding holes in the model that causes the reward to be over-predicted.

We then used an online data collection process identical to the one we used for phase 1 to jointly learn the embedded control space while updating the models so that the embedded controller doesn't find bad regions in them. We show where the DCEM updates fit into alg. 3. One alternative that we tried to updating the decoder to optimize the control cost on the samples from the replay buffer is that the decoder can also be immediately updated after planning at every step. This seemed nice since it didn't require any additional DCEM solves, but we found that the decoder became too biased during the episode as samples at consecutive timesteps have nearly identical information. For the hyper-parameters, we kept most of the DCEM hyper-parameters fixed throughout this phase to 100 samples, 10 elites, and a temperature $\tau = 1$. We ablated across 1) the number of DCEM iterations taken to be $\{3, 5, 10\}$, 2) deleting the replay buffer from phase 1 or not, and 3) re-initializing the model or not from phase 1.

Phase 3: Policy optimization into the controller. Finally now that we have a differentiable policy class induced by this differentiable controller we can do policy learning to fine-tune parts of it. We initially chose Proximal Policy Optimization (PPO) (Schulman et al., 2017) for this phase because we thought that it would be able to fine-tune the policy in a few iterations without requiring a good estimate of the value function, but this phase also ended up consuming many timesteps from the environment. Crucially in this phase, we do **not** do likelihood fitting at all, as our goal is to show that PPO can be used as another useful signal to update the parts of a controller — we did this to isolate the improvement from PPO but in practice we envision more unified algorithms that use both signals at the same time. Using the standard PPO hyper-parameters, we collect 10 episodes for each PPO training step and ablate across 1) the number of passes to make through these episodes $\{1, 2, 4\}$, 2) every combination of the reward, transition, and decoder being fine-tuned or frozen, 3) using a fixed variance of 0.1 around the output of the controller or learning this, 4) the learning rate of the fine-tuned model-based portions $\{10^{-4}, 10^{-5}\}$.

Algorithm 3 PlaNet (Hafner et al., 2018) variant that we use for proprioceptive control with optional DCEM embedding

▷ **Models:** a deterministic state model, a stochastic state model, a reward model, and (if using DCEM) an action sequence decoder.
▷ Initialize dataset $\mathcal{D}$ with S random seed episodes.
▷ Initialize the transition model's deterministic hidden state h_0 and initialize the environment, obtaining the initial state estimate x_0.
▷ CEM-Solve can use DCEM or full CEM
for $t = 1, \ldots, T$ **do**
  $u_t \leftarrow$ CEM-solve(h_{t-1}, x_{t-1})
  Add exploration noise $\epsilon \sim p(\epsilon)$ to the action u_t.
  $\{r_t, x_{t+1}, d_t\} \leftarrow$ env.step(u_t) ▷ Properly restarting if necessary
  Add $[r_t, x_t, u_t, d_t]$ to $\mathcal{D}$
  $h_t =$ update-hidden(h_{t-1}, x_t, u_t, d_t)
  if $t \equiv 0 \pmod{\text{update-interval}}$ **then**
    Sample trajectories $\tau = [r_\tau, x_\tau, u_\tau, d_\tau]_{\tau=1}^{H} \sim \mathcal{D}$ from the dataset.
    Obtain the hidden states of the $\{h_\tau, \hat{x}_\tau\}$ from the model.
    Compute the multi-step likelihood bound $\mathcal{L}(\tau, h_\tau, \hat{x}_\tau)$ ▷ Eq. 6 of Hafner et al. (2018)
    $\theta \leftarrow$ grad-update$(\nabla_\theta \mathcal{L}_\theta(\tau, h_\tau, \hat{x}_\tau))$ ▷ Optimize the likelihood bound
    if using DCEM **then**
      $\hat{z}_\tau = \arg\min_{z \in \mathcal{Z}} C_\theta(z; h_\tau, \hat{x}_\tau)$ ▷ Solve the embedded control problem in eq. (8)
      $\theta \leftarrow$ grad-update$(\nabla_\theta \sum_\tau C_\theta(\hat{z}_\tau))$ ▷ Update the decoder
    end if
  end if
end for

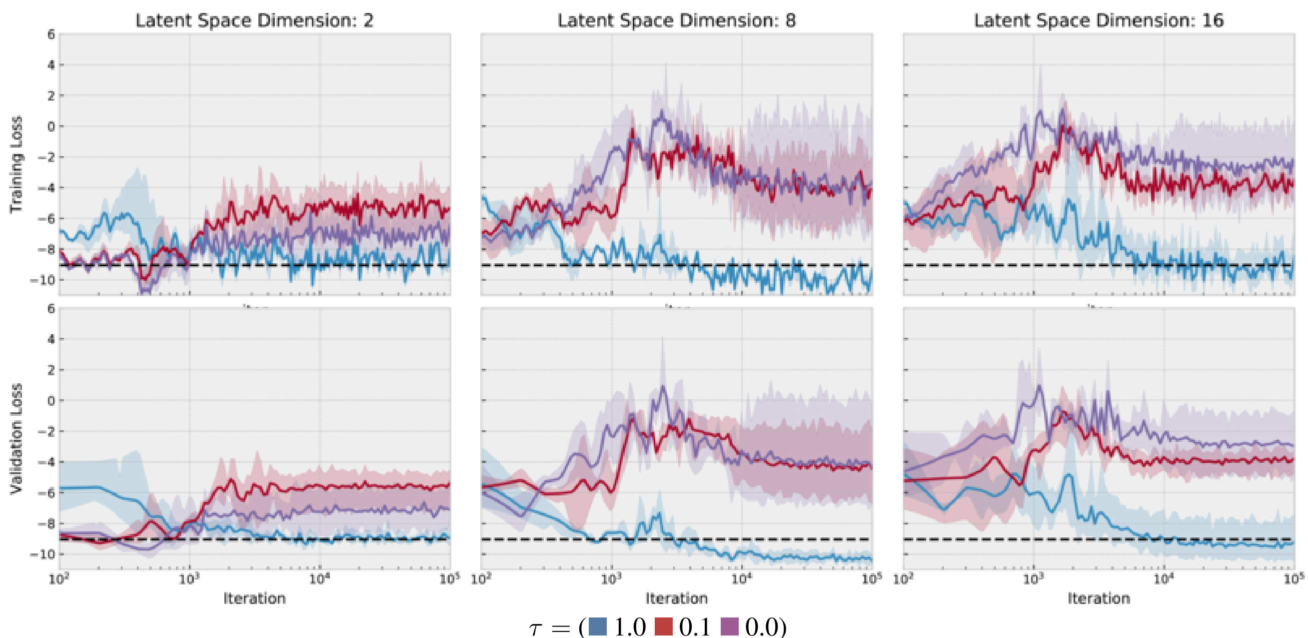

Figure 9. Training and validation loss convergence for the cartpole task. The dashed horizontal line shows the loss induced by an expert controller. Larger latent spaces seem harder to learn and as DCEM becomes less differentiable, the embedding is more difficult to learn. The shaded regions show the 95% confidence interval around three trials.

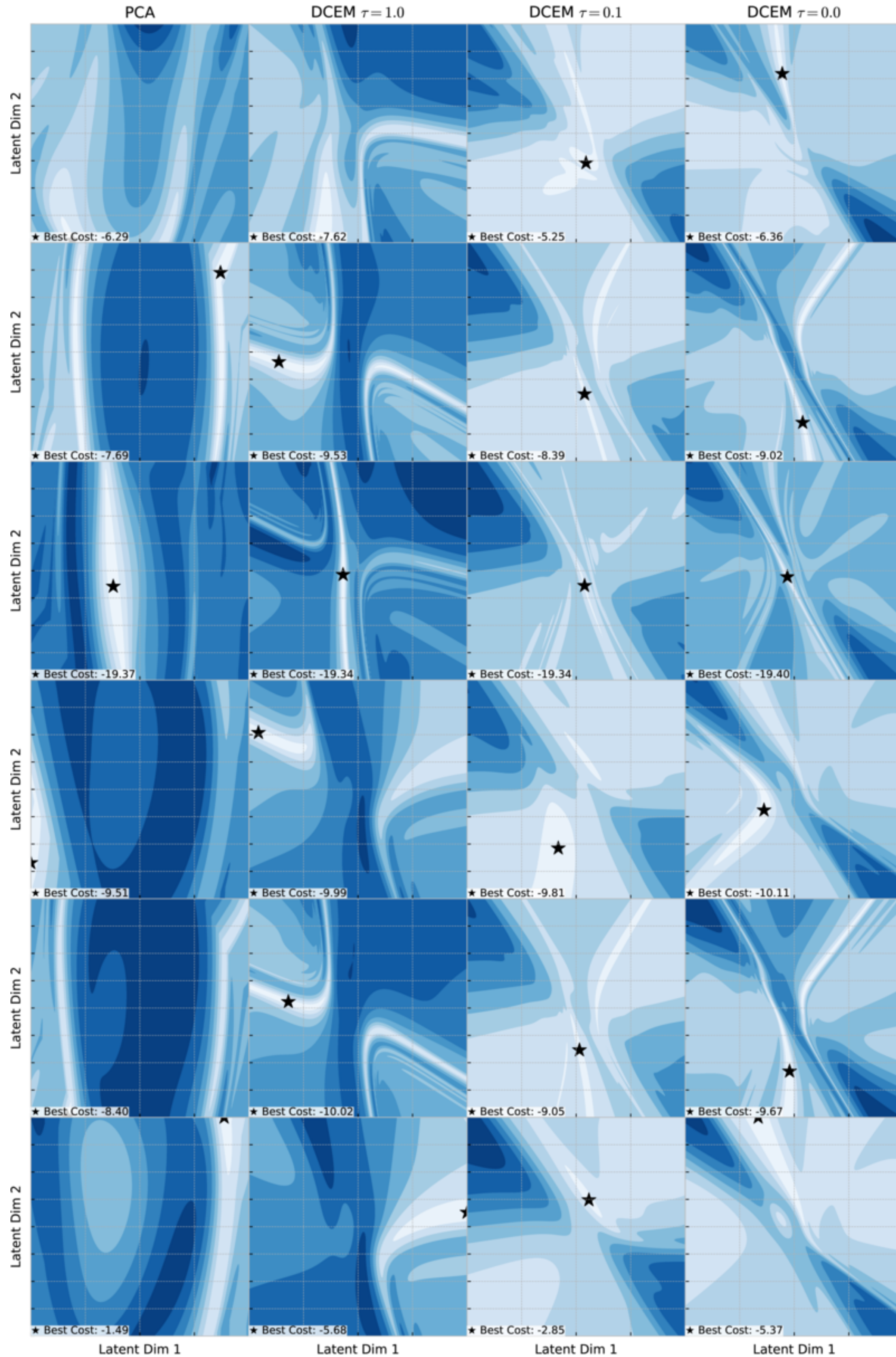

Figure 10. Learned DCEM reward surfaces for the cartpole task. Each row shows a different initial state of the system. We can see that as the temperature decreases, the latent representation can still capture near-optimal values, but they are in much narrower regions of the latent space than when $\tau = 1$.