A. Model Architecture and Training Details

We use the latent dynamics and reward models from PlaNet (Hafner et al., 2019) with default hyperparameters. We set the image size to 64x64. For every $N = 1$ episode collected, we train for $\text{It} = 15$ iterations. All models are trained on a single high-end GPU.

B. Planning Details

Algorithm 2 Gaussian LatCo

```
 1: Start with any available data $\mathcal{D}$
 2: while not converged do
 3:   for each time step $t = 1 \ldots T_{\text{tot}}$ with step $T_{\text{cache}}$ do
 4:     Infer latent state: $z_t \sim q(z_t|o_t)$
 5:     Define the Lagrangian:
          $$\mathcal{L}(\mu_{t+1:t+H}, \sigma_{t+1:t+H}, a_{t:t+H}, \lambda) = \sum_t \Big[ \mathbb{E}_{q(z_t)}[r(z_t)] - \lambda_t^{\text{dyn}}(||\text{mean}[p(z_t)] - \mu_t||^2 - \epsilon) - \lambda_t^{\text{dyn}}(||\text{stddev}[p(z_t)] - \sigma_t||^2 - \epsilon) - \lambda_t^{\text{act}}(\max(0, |a_t| - a_m)^2 - \epsilon^{\text{act}}) \Big]. \quad (11)$$
 6:     for each optimization step $k = 1 \ldots K$ do
 7:       Update plan:
          $\mu_{t+1:t+H}, \sigma_{t+1:t+H}, a_{t:t+H} \mathrel{+}= \tilde{\nabla}\mathcal{L}$    ▷ Eq (9)
 8:       Update dual variables:
          $\lambda_{t:t+H} := \textsc{Update}(\mathcal{L}, \lambda_{t:t+H})$    ▷ Eq (10)
 9:     end for
10:     Execute $a_{t:t+T_{\text{cache}}}$ in environment:
        $o_{t:t+T_{\text{cache}}}, r_{t:t+T_{\text{cache}}} \sim p_{env}$
11:   end for
12:   Add episode to replay buffer:
      $\mathcal{D} := \mathcal{D} \cup (o_{1:T_{\text{tot}}}, a_{1:T_{\text{tot}}}, r_{1:T_{\text{tot}}})$
13:   for training iteration $i = 1 \ldots \text{It}$ do
14:     Sample minibatch from replay buffer:
        $(o_{1:T}, a_{1:T}, r_{1:T})_{1:b} \sim \mathcal{D}$
15:     Train dynamics model:
        $\phi \mathrel{+}= \alpha\nabla\mathcal{L}_{\text{ELBO}}(o_{1:T}, a_{1:T}, r_{1:T})_{1:b}$    ▷ Eq (2)
16:   end for
17: end while
```

CEM & MPPI. On Metaworld tasks, we optimize for 100 iterations, where in each iteration, 10000 action sequences are sampled and the distribution is refit to the 100 best samples. On DM Control tasks, we optimize for 10 steps, where in each iteration, 1000 action sequences are sampled and the distribution is refit to the 100 best samples. The MPPI shares most hyperparameters with CEM, with the additional parameter $\gamma = 10$. For MPPI, we observed that

Algorithm 3 Gaussian LatCo Lagrangian computation

```
1: Given: initialized plan $\mu_{t+1:t+H}, \sigma_{t+1:t+H}, a_{t:t+H}$.
2: for time $t$ within planning horizon do
3:   Sample $K = 50$ latent states from the plan distribution: $z_t^k \sim q(z_t)$
4:   Evaluate the reward term with samples: $\mathbb{E}_{q(z_t)} r(z_t) \approx \sum_k r(z_t^k)$
5:   Approximate the one-step prediction distribution with samples: $p(z_t) \approx \{z_{t+1}^k\}_{k=1..K}$, where $z_{t+1}^k \sim p(z_{t+1}|z_t^k, a_t)$.
6:   Evaluate the sample mean of the one-step prediction distribution: $\text{mean}[p(z_{t+1})] \approx \frac{1}{K}\sum_k z_{t+1}^k$.
7:   Evaluate the sample standard deviation of the one-step prediction distribution: $\text{stddev}[p(z_{t+1})] \approx \sqrt{\frac{1}{K}\sum_k (z_{t+1}^k - \text{mean}[p(z_{t+1})])^2}$.
8: end for
9: Define the Lagrangian:
     $$\mathcal{L}(\mu_{t+1:t+H}, \sigma_{t+1:t+H}, a_{t:t+H}, \lambda) = \sum_t \Big[ \mathbb{E}_{q(z_t)}[r(z_t)] - \lambda_t^{\text{dyn}}(||\text{mean}[p(z_t)] - \mu_t||^2 - \epsilon) - \lambda_t^{\text{dyn}}(||\text{stddev}[p(z_t)] - \sigma_t||^2 - \epsilon) - \lambda_t^{\text{act}}(\max(0, |a_t| - a_m)^2 - \epsilon^{\text{act}}) \Big]. \quad (12)$$
```

both 10000 and 1000 action sequences yield similar results, so we use 1000 in our final results. We have manually tuned these hyperparameters, and we report the best results.

GD. On Metaworld tasks, we optimize for 500 iterations using the Adam optimizer (Kingma & Ba, 2015) (which is a modified version of momentum gradient descent) with learning rate 0.05. We use dual descent to penalize infeasible action predictions. The Lagrange multipliers are updated every 5 optimization steps. On DM Control tasks, all hyperparameters are the same except that we optimize for only 100 iterations. We have manually tuned the learning rate and tried several first-order optimizers, and we report the best results.

LatCo. On Metaworld tasks, we optimize for 200 iterations using the Levenberg-Marquardt optimizer with damping 10^{-3}. The damping parameter controls the trust region, with smaller or zero damping speeding up convergence, but potentially leading to numerical instability or divergence. The Lagrange multipliers are updated every step using the rule from Section 4, with $\epsilon^{\text{dyn}} = \epsilon^{\text{act}} = 10^{-4}$ and $\eta = 0.01$. The threshold ϵ directly controls the magnitude of the final dynamics and action violations. In general, we found

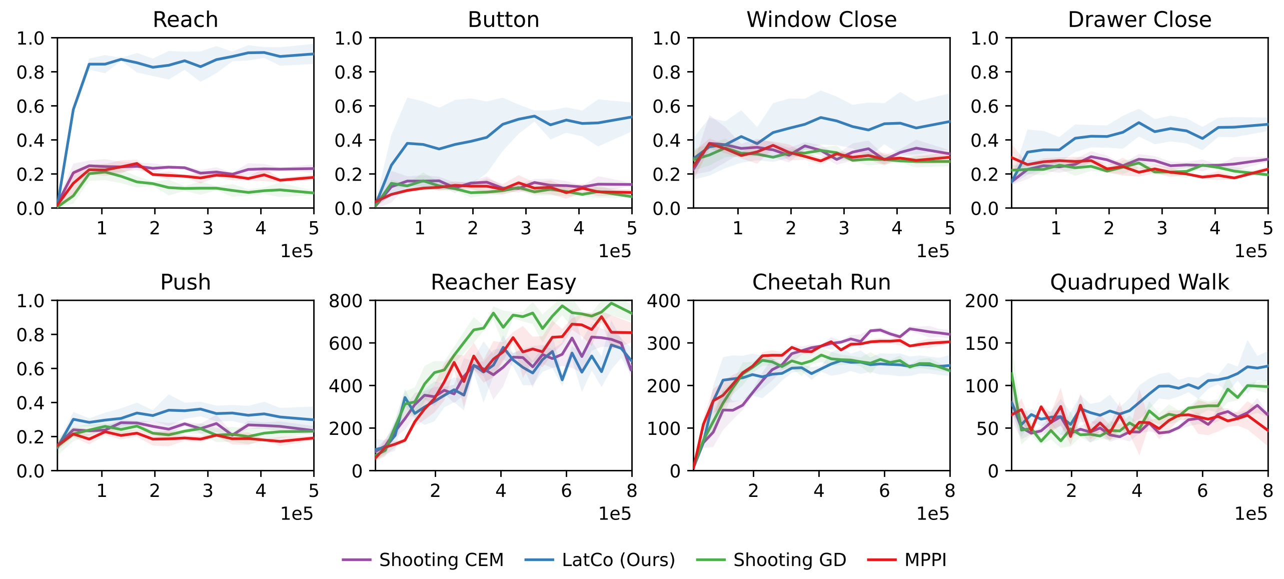

Figure 8: Learning curves for online MBRL experiments.

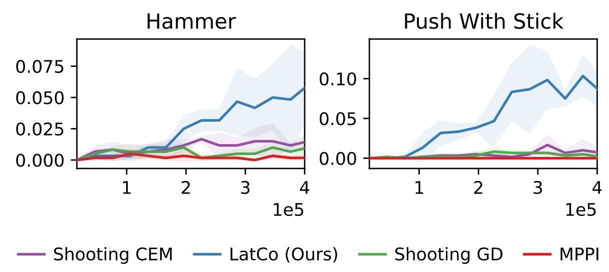

Figure 9: Learning curves for MBRL experiments with offline and online data.

this parameter to be most important for good performance, as a large threshold may cause infeasible plans, while low threshold would make the initial relaxation of the dynamics constraint less effective. We observed that a single threshold of 10^{-4} works for all of our Metaworld environments. η controls the update of the Lagrange multipliers. A larger η makes the optimization more aggressive but less stable, and a smaller η diminishes the effect of multiplier updates. We initilalize $\lambda_0^{\text{dyn}} = 1, \lambda_0^{\text{act}} = 1$. On DM Control tasks, all hyperparamers are the same except that we optimize for 100 iterations and set $\epsilon^{\text{dyn}} = \epsilon^{\text{act}} = 10^{-2}$.

Gaussian LatCo. The hyperparameters for Gaussian LatCo are largely the same to deterministic LatCo. However, we observed that Gaussian LatCo requires less optimization steps to converge and use 50 iterations. Further, since the threshold ϵ^{dyn} is now used for both the mean and the variance, it is setto 10^{-2}

LatCo no relaxation. We prevent the relaxation of dynamics constraint by initializing the Lagrange multipliers to 10^8.

LatCo no constrained optimization. We manually tune the multiplier values and fix them to $\lambda^{\text{dyn}} = 8, \lambda^{\text{act}} = 16$ throughout optimization.

LatCo no second order. We use 5000 optimization steps and update the Lagrange multipliers every 5 steps instead of 1. Further, for this ablation we use the simple gradient descent update rule for lagrange multipliers with learning rate of 1.5.

Image Collocation. We use the same hyperparameters as the *no second-order opt* ablation, except the learning rate, which is set to 0.02.

These planning hyperparameters remain fixed across the experiments as we observe that reward optimization converges in all cases. Planning a 30-step trajectory takes 12, 14, and 14 seconds for CEM, GD, and LatCo respectively on a single RTX 2080Ti graphics card. The action limits a_m are set to the limits of the environment, in our case always $a_m = 1$.

C. Gaussian LatCo details

We provide a detailed algorithm for Gausssian LatCo in Algorithm 2 and a detailed algorithm for evaluating the Lagrangian in Algorithm 2. The gradients are estimated with reparametrization. We observed that optimization stability is improved by only optimizing the variance σ with next step gradients. That is, for each variance parameter σ_t we only optimize it with respect to terms of the Lagrangian

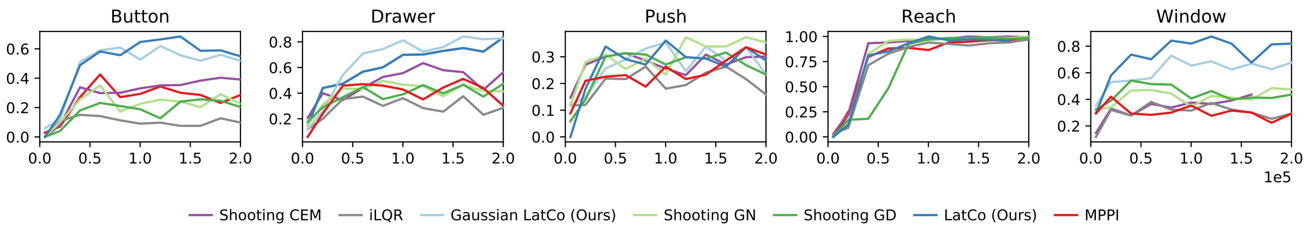

Figure 10: Online MBRL results on the Metaworld tasks with dense rewards. With dense reward, collocation still outperforms shooting methods, however, shooting methods show more progress, especially on the reaching task. We include an additional iLQR baseline implemented with our latent dynamics model. This method performs poorly, likely due to the extremely non-linear and high-dimensional latent space. Further, we include a comparison against Gaussian LatCo. We observe that Gaussian LatCo performs comparably to the deterministic version. We observed similar results from Gaussian LatCo on the sparse reward tasks, but are unable to include the full experiment due to computational requirements.

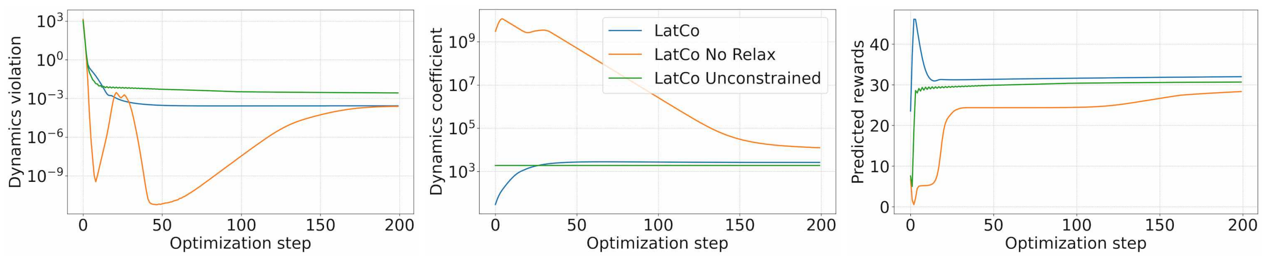

Figure 11: Additional optimization curves. The dynamics coefficient (magnitude of Lagrange multipliers) increases exponentially as the dynamics constraint is enforced, and eventually converges for LatCo. LatCo No Relax does not relax the dynamics initially as the coefficient is initialized at a high value. We see that this leads to suboptimal reward optimization as the found solution is not as good as with the full LatCo.

that also involve $q(z_{t-1})$, but we set the gradient of any term that involves $q(z_{t+1})$ to zero. This is similar to the approach of Patil et al. (2015) and corresponds to a more shooting-like formulation since the variance information is only propagated forward in the trajectory and not backward. Even though this is not the full gradient of the Lagrangian, we observed optimization still works well as the variance can always flexibly match its target.

D. Compared Methods and Ablations Details

CEM. The Cross-Entropy Method (CEM) is a sampling-based trajectory optimization technique. The method maintains a distribution of action sequence, initialized as an isotropic unit Gaussian. In each iteration, the method samples n action sequences from the current distribution and forwarded them through the model to obtain their predicted rewards. It then re-fits the distribution to the top k trajectories with the highest rewards by computing their empirical mean and standard deviation.

MPPI. MPPI is a variant of CEM where instead of taking the average of the elite samples, it takes the weighted average when estimating the mean and variance of the new distribution. The weights are computed by taking the Softmax of the rewards with a temperature parameter γ. The temperature parameter effectively controls the width of the distribution.

Shooting GD. This method optimizes a single action trajectory. The trajectory is initialized from a uniform distribution and optimized with gradient descent, with the objective being negative reward.

Shooting GN. This method is similar to Shooting GD, but it use a Gauss-Newton (GN) optimizer instead of gradient descent. The Gauss-Newton optimizer in this case is easy to implement and has fast runtime since the number of optimized variables is small for shooting.

iLQR. We additionally implemented an iLQR baseline with the same latent dynamics model as used in our method. We largely follow Tassa et al. (2012) for our implementation. Specifically, we implement line search on the optimization step size, and multiply the trust-region regularizer by 2 when the Q_{uu} matrix fails to be positive definite. We observed that using feedback controllers leads for worse results and only use the actions planned during optimization.

Image collocation. We directly optimize images instead of latent states using the same RSSM model. Images do not constitute a markov space, therefore we optimize them recurrently. Specifically, we evaluate the reward of image

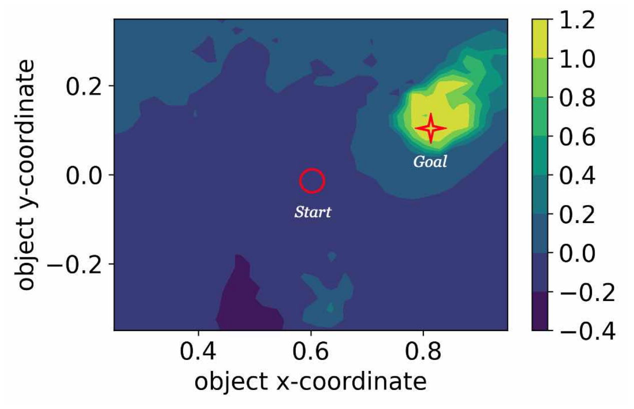

Figure 12: Visualization of the reward predictor for the Sawyer Pushing task. The output of the reward predictor is shown for each object position on the 2D table. We see that the reward predictor correctly predicts a value of 1 at the goal, and low values otherwise. In addition, there is a certain amount of smoothing induced by the reward predictor, which creates a gradient from the start to the goal position. This explains why gradient-based planning is applicable even in this sparse reward task. We note that this reward smoothing is caused simply by the fact we are training a neural reward predictor, and does not require any additional setup.

sequence by encoding them into the latent space and predicting the reward. We evaluate the dynamics constraint on the frame I_{t+1} by encoding the past sequence $I_{1:t}$ into the latent space, rolling out a one-step prediction, and decoding the images. Further, we observed poor optimization performance when optimizing recurrent constraints. Instead, we treat all constraints as pairwise by setting the gradients on $I_{1:t-1}$ to zero. We use gradient descent to optimize Image Collocation.

E. Additional experimental results

E.1. Learning Curves

We provide the learning curves for results in Figures 8 and 9. The online agents were trained for 500K environment steps on the MetaWorld tasks and 800K environment steps on the DM Control tasks. The hybrid agents were trained for 400K environment steps.

E.2. Dense MetaWorld tasks and Gaussian LatCo results

We additionally evaluated all methods on the MetaWorld tasks with their original shaped rewards. We observed that the softplus reward transformation used by LatCo tends to squash large positive rewards, resulting in ineffective optimization of the reward objective. Thus, we keep the running mean and standard deviation of rewards during training, and normalize the environment rewards with these metrics. This keeps the reward magnitude in a reasonable range and facilitates LatCo training. Learning curves for the dense reward tasks are shown in Fig. 10.

E.3. Analysing optimization

We visualize the additional optimization curves in Fig 11.

E.4. Analysing sparse reward planning

We observed that our method is able to solve sparse reward tasks using gradient-based planning. While this may be surprising at first, similar observations were made by prior work (Singh et al., 2019). We visualize the reward predictor output on the Pushing task in Fig 12.