A Proofs

A.1 Proof for Proposition 1

For each k, we denote the stationary point of gradient descent as $x_k^* = \lim_{n\to\infty} x_k^n$. Using the first-order optimality condition at the stationary point, we know that

$$\begin{aligned} 0 &= \nabla_x d_{\sigma_k}(x;\mathcal{D})^2\big|_{x=x_k^*} \\ &= -\nabla_x \sigma_k^2 \log\left[\sum_i \exp\left[-\tfrac{1}{2\sigma_k^2}\|x-x_i\|^2\right]\right]\Bigg|_{x=x_k^*} \\ &= \frac{\sigma_k^2 \sum_i \exp\left[-\frac{1}{2\sigma_k^2}\|x-x_i\|^2\right]\sigma_k^{-2}(x-x_i)}{\sum_i \exp\left[-\frac{1}{2\sigma_k^2}\|x-x_i\|^2\right]}\Bigg|_{x=x_k^*}. \end{aligned} \tag{13}$$

We then have that

$$\begin{aligned} \sum_i \exp\left[-\tfrac{1}{2\sigma_k^2}\|x_k^*-x_i\|^2\right](x_k^*-x_i) &= 0 \\ (\sum_i \exp\left[-\tfrac{1}{2\sigma_k^2}\|x_k^*-x_i\|^2\right])x_k^* &= \sum_i \exp\left[-\tfrac{1}{2\sigma_k^2}\|x_k^*-x_i\|^2\right]x_i \\ x_k^* &= \sum_i \frac{\exp\left[-\frac{1}{2\sigma_k^2}\|x_k^*-x_i\|^2\right]}{\sum_n \exp\left[-\frac{1}{2\sigma_k^2}\|x_k^*-x_n\|^2\right]} x_i. \end{aligned} \tag{14}$$

Let $x_j \in \operatorname{argmin}_{x_i\in\mathcal{D}} \|x_k^*-x_i\|^2$ and $\Delta_i(\sigma_k) = \sigma_k^{-2}(\|x_k^*-x_i\|^2 - \|x_k^*-x_j\|^2)$.

Lemma 4. *When there is a unique minimizer x_j, $\lim_{k\to\infty} x_k^* = x_j$.*

Proof. Since x_j is the unique closest data point to x_k^*, $\Delta_i > 0$ for $i \neq j$ and $\Delta_j = 0$. With the monotonically decreasing $\sigma_k \to 0$, we know that

$$\lim_{k\to\infty} \Delta_i(\sigma_k) = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{otherwise.} \end{cases} \tag{15}$$

Therefore,

$$\lim_{k\to\infty} \exp\left[-\tfrac{1}{2}\Delta_i(\sigma_k)\right] = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

We then obtain

$$\begin{aligned} \lim_{k\to\infty} \frac{\exp\left[-\frac{1}{2\sigma_k^2}(\|x_k^*-x_i\|^2 - \|x_k^*-x_j\|^2)\right]}{\sum_n \exp\left[-\frac{1}{2\sigma_k^2}(\|x_k^*-x_n\|^2 - \|x_k^*-x_j\|^2)\right]} &= \lim_{k\to\infty} \frac{\exp\left[-\frac{1}{2}\Delta_i(\sigma_k)\right]}{\sum_{n\neq j}\exp\left[-\frac{1}{2}\Delta_n(\sigma_k)\right]+1} \\ &= \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{17}$$

Therefore, by combining (14) and (17), we have that

$$\lim_{k\to\infty} x_k^* = x_j. \tag{18}$$

□

Lemma 5. *When there are multiple minimizers $x_{j_1},\ldots,x_{j_m} \in \operatorname{argmin}_{x_i\in\mathcal{D}} \|x_k^*-x_i\|^2$, $\lim_{k\to\infty} x_k^* = \frac{1}{m}\sum_{l=1}^m x_{j_l}$.*

Proof. In contrast to (17) for the unique x_j, we have that

$$\begin{aligned}
&\lim_{k\to\infty} \frac{\exp\left[-\frac{1}{2}(\|x_k^* - x_i\|^2 - \|x_k^* - x_{j_1}\|^2)\right]}{\sum_n \exp\left[-\frac{1}{2}(\|x_k^* - x_n\|^2 - \|x_k^* - x_{j_1}\|^2)\right]} \\
&= \lim_{k\to\infty} \frac{\exp\left[-\frac{1}{2}\Delta_i(\sigma_k)\right]}{\sum_{n\notin\{j_1,\ldots,j_m\}} \exp\left[-\frac{1}{2}\Delta_n(\sigma_k)\right] + m} \\
&= \begin{cases} \frac{1}{m} & \text{if } i \in \{j_1, \ldots, j_m\} \\ 0 & \text{otherwise.} \end{cases}
\end{aligned} \tag{19}$$

Therefore, x_k^* converges to the geometric center of all the minimizers:

$$\lim_{k\to\infty} x_k^* = \frac{1}{m}\sum_{l=1}^{m} x_{j_l}. \tag{20}$$

□

Lemma 6. *For a local minimizer x_k^*, x_j is unique; for a local maximizer or saddle point x_k^*, x_j is not unique.*

Proof. We first prove that x_j is unique if x_k^* is a local minimizer by contradiction. If there are multiple minimizers $x_{j_1}, \ldots, x_{j_m} \in \operatorname{argmin}_{x_i\in\mathcal{D}} \|x_k^* - x_i\|^2$, we have that

$$\begin{aligned}
\lim_{k\to\infty} d_{\sigma_k}(x_k^*; \mathcal{D})^2 &= \lim_{k\to\infty} -\sigma_k^2 \log\left[\sum_i \exp\left[-\tfrac{1}{2\sigma_k^2}\|x_k^* - x_i\|^2\right]\right] \\
&= -\log\left[\lim_{k\to\infty}\left(\sum_i \exp\left[-\tfrac{1}{2\sigma_k^2}\|x_k^* - x_i\|^2\right]\right)^{\sigma_k^2}\right] \\
&= \min_i \lim_{k\to\infty} \|x_k^* - x_i\|^2 \\
&= \lim_{k\to\infty} \|x_k^* - x_{j_1}\|^2 \\
&= \|\frac{1}{m}\sum_{l=1}^{m} x_{j_l} - x_{j_1}\|^2.
\end{aligned} \tag{21}$$

We observe that x_k^* is a local maximizer of $\min_x d_{\sigma_k}(x; \mathcal{D})^2$, which raises contradiction.

Similarly, we prove that x_j is not unique if x_k^* is a local maximizer or saddle point by contradiction. Assume x_j is the unique minimizer of $\min_{x_i\in\mathcal{D}} \|x_k^* - x_i\|^2$, we have

$$\begin{aligned}
\lim_{k\to\infty} d_{\sigma_k}(x_k^*; \mathcal{D})^2 &= \min_i \lim_{k\to\infty} \|x_k^* - x_i\|^2 \\
&= \lim_{k\to\infty} \|x_k^* - x_j\|^2 \\
&= 0.
\end{aligned} \tag{22}$$

Hence, x_k^* can not be a local maximizer or saddle point of $\min_x d_{\sigma_k}(x; \mathcal{D})^2$, which raises contradiction. □

Since gradient descent converges to a local minimizer almost surely with random initialization [69] and appropriate step sizes, there is a unique x_j for x_k^* and (18) holds. We also note that a similar conclusion can be reached by using Γ-convergence of the softmin to min as $\sigma_k \to 0$ [70].

A.2 Proof for Proposition 2

Let L_e be the local Lipschitz constant of the error $e(x) \coloneqq f(x) - f_\theta(x)$ over some domain $\mathcal{Z} \subseteq \mathcal{X}$, and define $x_c \coloneqq \arg\min_{x_i \in \mathcal{D}} \frac{1}{2}\|x - x_i\|_2^2$, i.e., the closest data-point. Then, we have the following:

$$
\begin{aligned}
\|f(x) - f_\theta(x)\| &\leq \min_{x_i \in \mathcal{D}} \left[e(x_i) + L_e\|x - x_i\|_2\right] \\
&\leq e(x_c) + L_e\|x - x_c\|_2 \\
&= e(x_c) + \sqrt{2}L_e\sqrt{\frac{1}{2}\|x - x_c\|_2^2} \\
&= e(x_c) + \sqrt{2}L_e\sqrt{\frac{1}{2}\min_{x_i \in \mathcal{D}}\|x - x_i\|_2^2} \\
&= e(x_c) + \sqrt{2}L_e\sqrt{\min_{x_i \in \mathcal{D}}\frac{1}{2}\|x - x_i\|_2^2} \\
&\leq e(x_c) + \sqrt{2}L_e\sqrt{-\sigma^2 \log\left(\sum_i \exp\left(-\frac{1}{2\sigma^2}\|x - x_i\|^2\right)\right) + \sigma^2 \log N} \\
&= e(x_c) + \sqrt{2}L_e\sqrt{d_\sigma(x;\mathcal{D})^2 + C_2}
\end{aligned}
\tag{23}
$$

where $C_2 \coloneqq \sigma^2 \log N - C$, and C is the constant defined in (4). In the second line, we use the fact that as x_c is a feasible solution to the minimization in the first line, it is an upper bound on the optimal value. In the sixth line, we have used the fact that for any vector $v = [v_1, \ldots, v_n]^\top \in \mathbb{R}^n$, $\min\{v_1, \ldots, v_n\} \leq -\frac{1}{t}\log\sum_{i=1}^n \exp(-tv_i) + \frac{\log n}{t}$ for some scaling t. In the final line, we have applied the definition of $d_\sigma(x;\mathcal{D})$ from (4).

A.3 Proof for Proposition 3

The perturbed data distribution can be written as a sum of Gaussians, since

$$
\begin{aligned}
p_\sigma(x') &\coloneqq \int \hat{p}(x)\mathcal{N}(x'; x, \sigma^2\mathbf{I})dx \\
&= \int \left[\frac{1}{N}\sum_i \delta(x_i)\right]\mathcal{N}(x'; x, \sigma^2\mathbf{I})dx \\
&= \frac{1}{N}\sum_i \int \delta(x_i)\mathcal{N}(x'; x, \sigma^2\mathbf{I}) \\
&= \frac{1}{N}\sum_i \mathcal{N}(x_i; x, \sigma^2\mathbf{I}) \\
&= \frac{1}{N}\sum_i \mathcal{N}(x; x_i, \sigma^2\mathbf{I})
\end{aligned}
\tag{24}
$$

Then we consider the negative log of the perturbed data distribution multiplied by σ^2,

$$
\begin{aligned}
-\sigma^2 \log p_\sigma(x) &= -\sigma^2 \log\left[\frac{1}{N}\sum_i \mathcal{N}(x; x_i, \sigma^2\mathbf{I})\right] \\
&= -\sigma^2 \log\left[\sum_i \mathcal{N}(x; x_i, \sigma^2\mathbf{I})\right] + \log N \\
&= -\sigma^2 \log\left[\frac{1}{\sqrt{(2\pi\sigma)^n}}\sum_i \exp\left[-\frac{1}{2\sigma^2}\|x - x_i\|^2\right]\right] + \sigma^2 \log N \\
&= -\sigma^2 \log\left[\sum_i \exp\left[-\frac{1}{2\sigma^2}\|x - x_i\|^2\right]\right] + \sigma^2 \log N + \sigma^2 \frac{n}{2}\log(2\pi\sigma) \\
&= -\sigma^2 \mathsf{LogSumExp}_i\left[-\frac{1}{2\sigma^2}\|x - x_i\|^2\right] + C(N, n, \mathbf{\Sigma}) \\
&= \mathsf{Softmin}_\sigma\left[\frac{1}{2}\|x - x_i\|^2\right] + C(N, n, \mathbf{\Sigma})
\end{aligned}
\tag{25}
$$

where we define $C(N, n, \mathbf{\Sigma}) \coloneqq \sigma^2(\log N + n/2\log(2\pi\sigma))$.

B Details of the Planning algorithm

B.1 Computation of Gradients

Recall that the gradient of $\log p_\sigma(x_i, u_i)$ cost w.r.t. the input variable u_j can be written as

$$\nabla_{u_j}\log p(x_i, u_i) = \nabla_{x_i}\log p(x_i, u_i)\mathbf{D}_{u_j}x_i + \nabla_{u_i}\log p(x_i, u_i)\mathbf{D}_{u_j}u_i \tag{26}$$

where $\mathbf{D}$ denotes the Jacobian. Writing the dependence on each variable more explicitly, we have

$$\nabla_{u_j}\log p(x_i(u_j), u_i(u_j)) = \nabla_{x_i}\log p(x_i(u_j), u_i(u_j))\mathbf{D}_{u_j}x_i(u_j) \tag{27}$$

$$+ \nabla_{u_i}\log p(x_i(u_j), u_i(u_j))\mathbf{D}_{u_j}u_i(u_j) \tag{28}$$

where we note that $\mathbf{D}_{u_j}u_i = 1$ if $i = j$ and 0 otherwise. As long as $i > j$, we also note that x_i has a dependence on u_j. Instead of computing this gradient explicitly, we first rollout the trajectory to compute $x_i(u_j), u_i(u_j)$, and compute the score function. Then we ask: which quantity do we need such that it gives us the above expression when differentiated w.r.t. u_j? We use the following quantity,

$$c_{ij} = s_x(x_i, u_i)x_i(u_j) + s_u(x_i, u_i)u_i(u_j) \tag{29}$$

where the score terms have been detached from the computation graph. Note that c_{ij} is a scalar and allows us to use reverse-mode automatic differentiation tools such as `pytorch` [71].

B.2 Noise-Annealing During Optimization

Additionally, we anneal the noise level during iterations of Adam. Given a sequence σ_k with K being the total number of annealing steps, we run Adam for max $\text{iter}_{\text{max}}/K$ iterations, then run it with the next noise level.

B.3 Connection to Diffuser

We first lift the dynamics constraint into a quadratic penalty and write the penalty as $\log p(x_{t+1}|x_t, u_t)$. This equivalence is seen by considering a case where we fix x_t, u_t and perturb x_{t+1} with a Gaussian noise of scale σ. If (x_t, u_t, x_{t+1}) is in the dataset, it obeys $x_{t+1} = f(x_t, u_t)$ under real-world dynamics f. This allows us to write

$$
\begin{aligned}
p_\sigma(x_{t+1}|x_t, u_t) &= \mathcal{N}(x_{t+1}|f(x_t, u_t), \sigma^2\mathbf{I}) \\
\log p_\sigma(x_{t+1}|x_t, u_t) &= -\frac{1}{2}\|x_{t+1} - f(x_t, u_t)\|^2 + C
\end{aligned}
\tag{30}
$$

where C is some constant that does not effect the objective. Then, we rewrite our objective using the factoring $p(x_t, u_t) = p(u_t|x)p(x_t)$. This allows us to rewrite the objective of Equation (11) as

$$\begin{aligned} &\sum_{t=1}^{T} r_t(x_t, u_t) + \beta \sum_{t=1}^{T} \log p(u_t|x_t) + \beta \sum_{t=1}^{T} \log p(x_t) + \beta \sum_{t=1}^{T} \log p(x_{t+1}|x_t, u_t) \\ =&V(x_{1:T}, u_{1:T}) + \beta \log p(x_{1:T}, u_{1:T}) + \sum_{t=1}^{T} \log p(x), \end{aligned} \tag{31}$$

where the first two terms are the objectives in Diffuser [25].

B.4 First-Order Policy Search

We note that our original method for gradient computation can easily be extended to the setting of feedback first-order policy search, where we define the uncertainty-penalized value function as

$$\begin{aligned} \max_{\alpha} \quad & \mathbb{E}_{x_1 \sim \rho} \left[\sum_{t=1}^{T} r_t(x_t, u_t) + \beta\sigma^2 \sum_{t=1}^{T} \log p_\sigma(x_t, u_t; \mathcal{D}) \right] \\ \text{s.t.} \quad & x_{t+1} = f_\theta(x_t, u_t), u_t = \pi_\alpha(x_t) \quad \forall t \in [1, T], \end{aligned} \tag{32}$$

where ρ is some distribution of initial conditions. We rewrite the objective with an explicit dependence on α, and use a Monte-Carlo estimator for the gradient of the stochastic objective,

$$\begin{aligned} &\nabla_\alpha \mathbb{E}_{x_1 \sim \rho} \left[\sum_{t=1}^{T} r_t(x_t(\alpha), u_t(\alpha)) + \beta\sigma^2 \sum_{t=1}^{T} \log p_\sigma(x_t(\alpha), u_t(\alpha); \mathcal{D}) \right] \\ =&\mathbb{E}_{x_1 \sim \rho} \nabla_\alpha \left[\sum_{t=1}^{T} r_t(x_t(\alpha), u_t(\alpha)) + \beta\sigma^2 \sum_{t=1}^{T} \log p_\sigma(x_t(\alpha), u_t(\alpha); \mathcal{D}) \right] \\ \approx&\frac{1}{N} \sum_{i=1}^{N} \nabla_\alpha \left[\sum_{t=1}^{T} r_t(x_t(\alpha), u_t(\alpha)) + \beta\sigma^2 \sum_{t=1}^{T} \log p_\sigma(x_t(\alpha), u_t(\alpha); \mathcal{D}) \quad \text{s.t.} \quad x_1 = x_i \sim \rho \right], \end{aligned} \tag{33}$$

where the last equation denotes that fixing the initial condition to x_i sampled from ρ, and N is the number of samples in the Monte-Carlo process. Since r_t and f_θ are differentiable, we can obtain the gradient

$$\nabla_\alpha \sum_{t=0}^{T} r_t(x_t(\alpha), u_t(\alpha)) \tag{34}$$

after rolling out the closed-loop system starting from x_i and using automatic differentiation w.r.t. policy parameters α. To compute the gradient w.r.t. the score function, we similarly use the chain rule,

and differentiate it w.r.t α, which lets us compute

$$\begin{aligned} \frac{\partial}{\partial\alpha} \left[\sum_{t=1}^{T} \log p_\sigma(x_t, u_t) \right] &= \sum_{t=1}^{T} \frac{\partial}{\partial\alpha} \log p_\sigma(x_t(\alpha), u_t(\alpha)) \\ &= \sum_{t=1}^{T} \frac{\partial}{\partial x_t} \log p_\sigma(x_t, u_t) \frac{\partial x_t}{\partial\alpha} + \frac{\partial}{\partial u_t} \log p_\sigma(x_t, u_t) \frac{\partial u_t}{\partial\alpha} \\ &= \sum_{t=1}^{T} s_x(x_t, u_t; \sigma) \frac{\partial x_t}{\partial\alpha} + s_u(x_t, u_t; \sigma) \frac{\partial u_t}{\partial\alpha} \end{aligned} \tag{35}$$

where the last term is obtained by differentiating

$$\sum_{t=1}^{T} s_x(x_t, u_t; \sigma) x_t + s_u(x_t, u_t; \sigma) u_t \tag{36}$$

after detaching s_x and s_u from the computation graph.

B.5 Imitation Learning

We give more intuition for why maximizing the state-action likelihood leads to imitation learning. If the empirical data comes from an expert demonstrator, maximizing data likelihood leads to minimization of cross entropy between the state-action pairs encountered during planning and the state-action occupation measure of the demonstration policy, which is estimated with the perturbed empirical distribution $p_\sigma(x_t, u_t)$,

$$\sum_t \log p_\sigma(x_t, u_t) = \sum_t \log p_\sigma(u_t|x_t) + \sum_t \log p_\sigma(x). \tag{37}$$

Note that the $\log p_\sigma(u_t|x_t)$ is identical to the Behavior Cloning (BC) objective, while $\log p_\sigma(x_t)$ drives future states of the plan closer to states in the dataset. We note that Adversarial Inverse Reinforcement Learning (AIRL) [34] minimizes a similar objective as ours [12].

C Experiment Details

C.1 Cartpole with Learned Dynamics

Environment. We use the cart-pole dynamics model in [58, chapter 3.2], with the cost function being

$$c_t(x_t, u_t) = \begin{cases} ||x_t - x_g||^2_{\mathbf{Q}} & \text{if } t = T \\ 0 & \text{else.} \end{cases} \tag{38}$$

$\mathbf{Q} = \text{diag}(1, 1, 0.1, 0.1)$. We choose the planning horizon $T = 60$.

Training. We randomly collected a dataset of size $N = 1,000,000$ within the red box region in the state space. The dynamics model is an MLP with 3 hidden layers of width $(64, 64, 32)$. For ensemble approach we use 6 different dynamics models, all with the same network structures.

We train a score function estimator, represented by an MLP with 4 hidden layers of width 1024. The network is trained form 400 epochs with a batch size of 2048.

Parameters During motion planning, for Adam optimizer we use a learning rate of 0.01. For CEM approach we use a population size of 10, with standard variance $\sigma = 0.05$, and we take the top 4 seeds to update the mean in the next iteration.

Data Distance Estimator. To train a data distance estimator, we introduce a function approximator $d_\eta : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}$ parametrized by η to predict the noise-dependent $\mathsf{Softmin}$ distance. The training objective is given by

$$\min_\eta \frac{1}{2}\mathbb{E}_{x\in\Omega,\sigma\in[0,\sigma_{max}]}\left[\frac{1}{\sigma^2}\left|d(x,\sigma) - \mathsf{Softmin}_{x_i\in\mathcal{D}}\frac{1}{2}\|x - x_i\|^2_{\sigma^{-2}\mathbf{I}}\right|\right] \tag{39}$$

where Ω is a large enough domain that covers the data distribution $\mathcal{D}$. For small datasets, it is possible to loop through all x_i in the dataset to compute this loss at every iteration. However, this training can get prohibitive as all of the training set needs to be considered to compute the loss, preventing batch training out of the training set.

C.2 D4RL Dataset

Environment. We directly use the D4RL dataset [43] Mujoco tasks [60] with 3 different environments of halfcheetah, walker2d, and hopper. We additionally use differnet sources of data with random, medium, and medium-expert.

Training. The dynamics and the score functions are both parametrized with MLP with 4 hidden layers of width 1024. The noise-conditioned score function is implemented by treating each level of noise σ_k as an integer token, that gets embedded into a 1024 vector and gets multiplied with the output of each layer. This acts similar to a masking of the weights depending on the level of noise. The D4RL environment does not provide us with a differentiable reward function, so we additionally train an estimator for the reward. We empirically saw that for score function estimation, wide shallow networks performed better. We train both instances for 1000 iterations with Adam, with a learning rate of $1e-3$ and batch size of 2048.

We additionally set a noise schedule to be a cosine schedule that anneals from $\sigma = 0.2$ to $\sigma = 0.01$ for 10 steps in the normalized space of x, u.

Parameters We used a range of βs between $1e^{-3}$ and $1e^{-1}$ depending on the environment, where in some cases it helped to be more reliant on reward, and in others it's desirable to rely on imitation. We use a MPC with $T = 5$ and optimize it for 50 iterations with an aggressive learning rate of $1e^{-1}$.

C.3 Pixel Single Integrator

Environment. In this environment, we have a 2D single integrator $f(x_t, u_t) = x_t + u_t$, $x_t \in \mathbb{R}^2, u_t \in \mathbb{R}^2$ as the underlying ground-truth dynamics; however, instead of raw states x_t, we observe a 32×32 grayscale image $y_t = h(x_t) \in \mathbb{R}^{32\times 32}$, which are top-down renderings of the robot. In these observations, the position of the robot is represented with a dot. Moreover, we assume that we do not directly assign the 2D control input u_t, but instead propose a 32×32 grayscale image $\hat{u}_t$, where the value of the 2D control action u_t is extracted from the image via a spatial average:

$$u_t = \sum_{(p_x,p_y)} g_u(p_x, p_y)\hat{u}_t(p_x, p_y), \tag{40}$$

where the sum loops over each pixel $(p_x, p_y) \in \{1, \dots, 32\}^2$, $y_t(p_x, p_y)$ refers to the intensity of the image at pixel (p_x, p_y), and $g_u : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}^2$ is a grid function mapping from pixel (p_x, p_y) to a corresponding control action. The control image $\hat{u}_t$ is normalized such that its overall intensity sums to 1. Given some goal x_g, the reward is set to be the $-c_t(x_t, u_t)$, where the cost $c_t(x_t, u_t)$ is

$$c_t(x_t, u_t) = \begin{cases} \|x_t - x_g\|^2_{\mathbf{Q}_t} + \|u_t\|^2_{\mathbf{R}} & \text{if } t = T \\ \|x_t - x_g\|^2_{\mathbf{Q}} + \|u_t\|^2_{\mathbf{R}} & \text{else,} \end{cases} \tag{41}$$

To evaluate this cost function for the planned sequence of image observations and control images, the states x_t are also extracted from the image observations through a similar spatial averaging:

$$x_t = \sum_{(p_x,p_y)} g_{\text{im}}(p_x, p_y) y_t(p_x, p_y), \tag{42}$$

where $g_{\text{im}} : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}^2$ is a grid function mapping from pixel (p_x, p_y) to a corresponding state.

In other words, we have running costs for the state and input, and a different terminal cost for the state. We set $\mathbf{R} = 6.5\mathbf{I}$, $\mathbf{Q} = 500\mathbf{I}$, and $\mathbf{Q}_d = 1000\mathbf{I}$, and plan with a horizon of $T = 15$.

Training. We collect a randomly collected dataset of size $N = 200,000$, with underlying 2D data sampled from $x_t \in [-1, 1]^2$ and $u_t \in [-0.2, 0.2]^2$. Both the dynamics and the score function estimator are represented as U-Nets [72], with the architecture coming from [73].

Parameters. We found that $\beta = 0.5$ is sufficient for the penalty parameter. Results are obtained within 1450 iterations with a learning rate of 0.03. In representing the noise-conditioned score function, we use 232 smoothing parameters $\{\sigma_k\}_{k=1}^{232}$, from $\sigma_1 = 50$ to $\sigma_{232} = 0.01$.

Baselines. For gradient-based planning with ensembles, we set $\beta = 0.5$ and use an ensemble of size 10. For CEM with ensembles, we set $\beta = 0.5$, with an ensemble of size 5 (we only used the first five networks in the original ensemble of size 10 due to RAM limitations).

C.4 Box Pushing with Marker Dynamics

Environment. We prepare a box-pushing environment where we assume that the box follows quasistatic dynamics, which allows us to treat the marker positions directly as state of the box that is bijective with its pose. We use 2D coordinates for each markers, and append the pusher position, also in 2D, resulting in $x_t \in \mathbb{R}^{12}$. The pusher is given a relative position command with a relatively large step size [68]. In addition, we give the robot knowledge of the pusher dynamics, $x_{t+1}^{\text{pusher}} = x_t^{\text{pusher}} + u_t$. The general goal of the task is to push the box and align the edge of the box with the blue tape line.

We formulate our cost as

$$c_t(x_t, u_t) = \begin{cases} \|x_t^{\text{marker}} - x_g^{\text{marker}}\|_{\mathbf{Q}_T}^2 & \text{if } t = T \\ \|u_t\|_{\mathbf{R}}^2 & \text{else,} \end{cases} \tag{43}$$

where $\mathbf{Q}_T = \mathbf{I}$ and $\mathbf{R} = 0.1\mathbf{I}$. In order to get x_g^{marker}, we place the box where we want the goal to be and measure the position of the markers.

Training. We collect 100 demonstration trajectories resulting in 750 pairs of (x_t, u_t, x_{t+1}). The dynamics and the score functions are learned with a MLP of 4 hidden layers with size 1024, with the noise-conditioned score estimator being trained similar to the D4RL dataset with multiplicative token embeddings. We train for 500 iterations with a batch size of 32.

We additionally set a noise schedule to be a cosine schedule that anneals from $\sigma = 0.2$ to $\sigma = 0.01$ for 10 steps.

Parameters. We observed that $\beta = 1e^{-2}$ performs well for all the examples, with a learning rate of 0.1 and 50 iterations. We found that a horizon of $T = 4$ was sufficient for our setup.