A Learning and Sampling

A.1 Deep generative modelling

A complete trajectory is denoted by

$$\zeta = \{s_0, a_0, s_1, a_1, \cdots, a_{T-1}, s_T\}, \tag{1}$$

where T is the maximum length of all observed trajectories. The joint distribution of state and action sequences can be factorized according to the causal assumptions in nMDP:

$$\begin{aligned} p_\theta(\zeta) &= p(s_0)p_\alpha(a_0|s_0)p_\beta(s_1|s_0, a_0)\cdots p_\alpha(a_{T-1}|s_{0:T-1})p_\beta(s_T|s_{T-1}, a_{T-1}) \\ &= p(s_0)\prod_{t=0}^{T-1} p_\alpha(a_t|s_{0:t})p_\beta(s_{t+1}|s_t, a_t), \end{aligned} \tag{2}$$

where $p_\alpha(a_t|s_{0:t-1})$ is the policy model with parameter α, $p_\beta(s_t|s_{t-1}, a_{t-1})$ is the transition model with parameter β, both of which are parameterized with neural networks, $\theta = (\alpha, \beta)$. $p(s_0)$ is the initial state distribution, which can be sampled as a black box.

The density families of policy and transition are consistent with the conventional setting of IRL [4]. The transition describes the predictable change in state space, which is often possible to express the random variable s_{t+1} as a deterministic variable $s_{t+1} = g_\beta(s_t, a_t, \epsilon)$, where ϵ is an auxiliary variable with independent marginal $p(\epsilon)$, and $g_\beta(.)$ is some vector-valued function parameterized by β. The policy accounts for bounded rationality as a Boltzmann distribution with state-action value as the unnormalized energy:

$$p_\alpha(a_t|s_{0:t}) = \frac{1}{Z(\alpha, s_{0:t})}\exp\left(f_\alpha(a_t; s_{0:t})\right), \tag{3}$$

where $f_\alpha(a_t; s_{0:t})$ is the negative energy, $Z(\alpha, s_{0:t}) = \int \exp(f_\alpha(a_t; s_{0:t}))da_t$ is the normalizing constant given the history $s_{0:t}$.

Since we can only observe state sequences, the aforementioned generative model can be understood as a sequential variant of LEBM [16], where the transition serves as the generator and the policy is a history-conditioned latent prior. The marginal distribution of state sequences and the posterior distribution of action sequences are:

$$p_\theta(s_{0:T}) = \int p_\theta(s_{0:T}, a_{0:T-1})da_{0:T-1}, \quad p_\theta(a_{0:T-1}|s_{0:T}) = \frac{p_\theta(s_{0:T}, a_{0:T-1})}{p_\theta(s_{0:T})}. \tag{4}$$

A.2 Maximum likelihood learning

We need to estimate $\theta = (\alpha, \beta)$. Suppose we observe training examples: $\{\xi^i\}, i = 1, 2, \cdots, n, \quad \xi^i = [s_0^i, s_1^i, ..., s_T^i]$. The log-likelihood function is:

$$L(\theta) = \sum_{i=1}^{n} \log p_\theta(\xi^i). \tag{5}$$

Denote posterior distribution of action sequence $p_\theta(a_{0:T-1}|s_{0:T})$ as $p_\theta(A|S)$ for convenience where A and S means the complete action and state sequences in a trajectory. The gradient of log-likelihood is:

$$\begin{aligned} \nabla_\theta \log p_\theta(\xi) &= \nabla_\theta \log p_\theta(s_0, s_1, \cdots, s_T) \\ &= \mathbb{E}_{p_\theta(A|S)}[\nabla_\theta \log p_\theta(s_0, s_1, \cdots, s_T)] \\ &= \mathbb{E}_{p_\theta(A|S)}[\nabla_\theta \log p_\theta(s_0, s_1, \cdots, s_T)] + \mathbb{E}_{p_\theta(A|S)}[\nabla_\theta \log p_\theta(A|S)] \\ &= \mathbb{E}_{p_\theta(A|S)}[\nabla_\theta \log p_\theta(s_0, a_0, s_1, a_1, \cdots, a_{T-1}, s_T)] \\ &= \mathbb{E}_{p_\theta(A|S)}[\nabla_\theta \log p(s_0)p_\alpha(a_0|s_0)\cdots p_\alpha(a_{T-1}|s_{0:T-1})p_\beta(s_T|s_{T-1}, a_{T-1})] \\ &= \mathbb{E}_{p_\theta(A|S)}[\nabla_\theta \sum_{t=0}^{T-1}(\log p_\alpha(a_t|s_{0:t}) + \log p_\beta(s_{t+1}|s_t, a_t))] \\ &= \mathbb{E}_{p_\theta(A|S)}[\sum_{t=0}^{T-1}(\underbrace{\nabla_\alpha \log p_\alpha(a_t|s_{0:t})}_{\text{policy/prior}} + \underbrace{\nabla_\beta \log p_\beta(s_{t+1}|s_t, a_t)}_{\text{transition}})], \end{aligned} \tag{6}$$

where the third equation is because of a simple identity $\mathrm{E}_{\pi_\theta(a)}\left[\nabla_\theta \log \pi_\theta(a)\right] = 0$ for any probability distribution $\pi_\theta(a)$. Applying this simple identiy, we also have:

$$\begin{aligned} 0 &= \mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\nabla_\alpha \log p_\alpha(a_t|s_{0:t})\right] \\ &= \mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\nabla_\alpha f_\alpha(a_t; s_{0:t}) - \nabla_\alpha \log Z(\alpha, s_{0:t})\right] \\ &= \mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\nabla_\alpha f_\alpha(a_t; s_{0:t})\right] - \nabla_\alpha \log Z(\alpha, s_{0:t}). \end{aligned} \tag{7}$$

Due to the normalizing constant $Z(\alpha, s_{0:t})$ in the energy-based prior p_α, the gradient for the policy term involves both posterior and prior samples:

$$\begin{aligned} \delta_{\alpha,t}(S) &= \mathbb{E}_{p_\theta(A|S)}\left[\nabla_\alpha \log p_\alpha(a_t|s_{0:t})\right] \\ &= \mathbb{E}_{p_\theta(A|S)}\left[\nabla_\alpha f_\alpha(a_t; s_{0:t}) - \nabla_\alpha \log Z(\alpha, s_{0:t})\right] \\ &= \mathbb{E}_{p_\theta(A|S)}\left[\nabla_\alpha f_\alpha(a_t; s_{0:t}) - \mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\nabla f_\alpha(a_t; s_{0:t})\right]\right] \\ &= \mathbb{E}_{p_\theta(A|S)}\left[\nabla_\alpha f_\alpha(a_t; s_{0:t})\right] - \mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\nabla_\alpha f_\alpha(a_t; s_{0:t})\right], \end{aligned} \tag{8}$$

where $\delta_{\alpha,t}(S)$ denotes the surrogate loss of policy term for time step t. Intuition can be gained from the perspective of adversarial training [34, 35]: On one hand, the model utilizes action samples from the posterior $p_\theta(A|S)$ as pseudo-labels to supervise the unnormalized prior at each step $p_\alpha(a_t|s_{0:t})$. On the other hand, it discourages action samples directly sampled from the prior. The model converges when prior samples and posterior samples are indistinguishable.

To ensure the transition model's validity, it needs to be grounded in real-world dynamics when jointly learned with the policy. Otherwise, the agent would be purely hallucinating based on the demonstrations. Throughout the training process, we allow the agent to periodically collect self-interaction data with $p_\alpha(a_t|s_{0:t})$ and mix transition data from two sources with weight w_β:

$$\delta_{\beta,t}(S) = w_\beta \mathbb{E}_{p_\theta(A|S)}\left[\nabla_\beta \log p_\beta(s_{t+1}|s_t, a_t)\right] + (1 - w_\beta)\mathbb{E}_{p_\alpha(a_t|s_{0:t}),Tr}\left[\nabla_\beta \log p_\beta(s_{t+1}|s_t, a_t)\right]. \tag{9}$$

A.3 General transition model

We need to compute the gradient of β for the logarithm of transition probability in Equation 9, and as we will see in section 3.3, we also need to compute the gradient of the action during sampling actions. The reparameterization [43] is useful since it can be used to rewrite an expectation w.r.t $p_\beta(s_{t+1}|s_t, a_t)$ such that the Monte Carlo estimate of the expectation is differentiable, so we use delta function $\delta(.)$ to rewrite probability as an expectation:

$$\begin{aligned} p_\beta(s_{t+1}|s_t, a_t) &= \int \delta(s_{t+1} - s'_{t+1}) p_\beta(s'_{t+1}|s_t, a_t) ds'_{t+1} \\ &= \int \delta(s_{t+1} - g_\beta(s_t, a_t, \epsilon)) p(\epsilon) d\epsilon. \end{aligned} \tag{10}$$

Taking advantage of the properties of $\delta(.)$:

$$\int f(x)\delta(x)dx = f(0), \quad \delta(f(x)) = \Sigma_n \frac{1}{|f'(x_n)|}\delta(x - x_n), \tag{11}$$

where f is differentiable and have isolated zeros, which is x_n, we can rewrite the transition probability as:

$$\begin{aligned} p_\beta(s_{t+1}|s_t, a_t) &= \int \sum_n \frac{1}{|\frac{\partial}{\partial \epsilon} g_\beta(s_t, a_t, \epsilon)|_{\epsilon=\epsilon_n}} \delta(\epsilon - \epsilon_n) p(\epsilon) d\epsilon \\ &= \sum_n \frac{p(\epsilon_n)}{|\frac{\partial}{\partial \epsilon} g_\beta(s_t, a_t, \epsilon)|_{\epsilon=\epsilon_n}}, \end{aligned} \tag{12}$$

where ϵ_n is the zero of $s_{t+1} = g_\beta(s_t, a_t, \epsilon)$. Therefore, if we have a differentiable simulator $\nabla_{a_t} \log p_\beta(s_{t+1}|s_t, a_t)$ and the analytical form of $p(\epsilon)$, then gradient of both a_t and β for $\log p_\beta(s_{t+1}|s_t, a_t)$ can be computed.

The simplest situation is:

$$s_{t+1} = g_\beta(s_t, a_t) + \epsilon, \epsilon \sim p(\epsilon) = \mathcal{N}(0, \sigma^2). \tag{13}$$

In this case, there is only one zero ϵ^* for the transition function, $s_{t+1} = g_\beta(s_t, a_t) + \epsilon^*$, and the gradient of log probability is:

$$\begin{aligned}\nabla \log p_\beta(s_{t+1}|s_t, a_t) &= \nabla \log \frac{p(\epsilon^*)}{|\frac{\partial}{\partial \epsilon}(g_\beta(s_t, a_t) + \epsilon)|_{\epsilon=\epsilon^*}} \\ &= \nabla \log p(\epsilon^*) \\ &= \nabla \log p(s_{t+1} - g_\beta(s_t, a_t)) \\ &= \frac{1}{\sigma^2}(s_{t+1} - g_\beta(s_t, a_t))\nabla g_\beta(s_t, a_t).\end{aligned} \tag{14}$$

A.4 Prior and posterior sampling

The maximum likelihood estimation requires samples from the prior and the posterior distributions of actions. It would not be a problem if the action space is quantized. However, since we target general latent action learning, we proceed to introduce sampling techniques for continuous actions.

When sampling from a continuous energy space, short-run Langevin dynamics [17] can be an efficient choice. For a target distribution $\pi(a)$, Langevin dynamics iterates $a_{k+1} = a_k + s\nabla_{a_k} \log \pi(a_k) + \sqrt{2s}\epsilon_k$, where k indexes the number of iteration, s is a small step size, and ϵ_k is the Gaussian white noise. $\pi(a)$ can be either the prior $p_\alpha(a_t|s_{0:t})$ or the posterior $p_\theta(A|S)$. One property of Langevin dynamics that is particularly amenable for EBM is that we can get rid of the normalizing constant. So for each t the iterative update for prior samples is

$$a_{t,k+1} = a_{t,k} + s\nabla_{a_{t,k}} f_\alpha(a_{t,k}; s_{0:t}) + \sqrt{2s}\epsilon_k. \tag{15}$$

Given a state sequence $s_{0:T}$ from the demonstrations, the posterior samples at each time step a_t come from the conditional distribution $p(a_t|s_{0:T})$. Notice that with Markov transition, we can derive

$$p_\theta(a_{0:T-1}|s_{0:T}) = \prod_{t=0}^{T-1} p_\theta(a_t|s_{0:T}) = \prod_{t=0}^{T-1} p_\theta(a_t|s_{0:t+1}). \tag{16}$$

The point is, given the previous and the next subsequent state, the posterior can be sampled at each step independently. So the posterior iterative update is

$$\begin{aligned}a_{t,k+1} &= a_{t,k} + s\nabla_{a_{t,k}} \log p_\theta(a_{t,k}|s_{0:t+1}) + \sqrt{2s}\epsilon_k \\ &= a_{t,k} + s\nabla_{a_{t,k}} \log p_\theta(s_{0:t}, a_{t,k}, s_{t+1}) + \sqrt{2s}\epsilon_k \\ &= a_{t,k} + s\nabla_{a_{t,k}} (\underbrace{\log p_\alpha(a_{t,k}|s_{0:t})}_{\text{policy/prior}} + \underbrace{\log p_\beta(s_{t+1}|s_t, a_t)}_{\text{transition}}) + \sqrt{2s}\epsilon_k.\end{aligned} \tag{17}$$

Intuitively, action samples at each step are updated with the energy of all subsequent actions and a single-step forward by back-propagation. However, while gradients from the transition term are analogous to the inverse dynamics in BCO [37], it may lead to poor training performance due to non-injectiveness in forward dynamics [38].

We develop an alternative posterior sampling method with importance sampling to overcome this challenge. Leveraging the learned transition, we have

$$p_\theta(a_t|s_{0:t+1}) = \frac{p_\beta(s_{t+1}|s_t, a_t)}{\mathbb{E}_{p_\alpha(a_t|s_{0:t})}[p_\beta(s_{t+1}|s_t, a_t)]} p_\alpha(a_t|s_{0:t}). \tag{18}$$

Let $c(a_t; s_{0:t+1}) = \mathbb{E}_{p_\alpha(a_t|s_{0:t})}[p_\beta(s_{t+1}|s_t, a_t)]$, posterior sampling from $p_\theta(a_{0:T-1}|s_{0:T})$ can be realized by adjusting importance weights of independent samples from the prior $p_\alpha(a_t|s_{0:t})$, in which the estimation of weights involves another prior sampling. In this way, we avoid back-propagating through non-injective dynamics and save some computation overhead in Eq. (17).

To train the policy, Eq. (8) can now be rewritten as

$$\delta_{\alpha,t}(S) = \mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\frac{p_\beta(s_{t+1}|s_t, a_t)}{c(a_t; s_{0:t+1})}\nabla_\alpha f_\alpha(a_t; s_{0:t})\right] - \mathbb{E}_{p_\theta(a_t|s_{0:t})}[\nabla_\alpha f_\alpha(a_t; s_{0:t})]. \tag{19}$$

A.5 Algorithm

The learning and sampling algorithms with MCMC and with importance sampling for posterior sampling are described in Algorithm 1 and Algorithm 2.

Algorithm 1: LanMDP without importance sampling

```
Input: Learning iterations $N$, learning rate for energy-based policy $\eta_\alpha$, learning rate for transition model $\eta_\beta$, initial parameters $\theta_0 = (\alpha_0, \beta_0)$, expert demonstrations $\{s_{0:H}\}$, context length $L$, batch size $m$, number of prior and posterior sampling steps $\{K_0, K_1\}$, prior and posterior sampling step sizes $\{s_0, s_1\}$.
Output: $\theta_N = (\alpha_N, \beta_N)$.
Reorganize $\{s_{0:H}\}$ to to state sequenec segments $(s_{t-L+1}, \cdots, s_{t+1})$ with length $L + 1$.
Use energy-based policy with $\alpha_0$ collect transitions to fill in the replay buffer.
Use transitions in replay buffer to pre-train transition model $\beta_0$.
for $t = 0$ to $N - 1$ do
    Demo sampling Sample observed examples $(s_{t-L+1}, \cdots, s_{t+1})_{i=1}^{m}$.
    Posterior sampling: Sample $\{a_t\}_{i=1}^{m}$ using Eq. (17) with $K_1$ iterations and stepsize $s_1$.
    Prior sampling: Sample $\{\hat{a}_t\}_{i=1}^{m}$ using Eq. (15) with $K_0$ iterations and stepsize $s_0$.
    Policy learning: Update $\alpha_t$ to $\alpha_{t+1}$ by Eq. (8) with learning rate $\eta_\alpha$.
    Transition learning: Update replay buffer with trajectories from current policy model $\alpha_{t+1}$, then update $\beta_t$ to $\beta_{t+1}$ by Eq. (9) with learning rate $\eta_\beta$.
end for
```

Algorithm 2: LanMDP with importance sampling

```
Input: Learning iterations $N$, learning rate for energy-based policy $\eta_\alpha$, learning rate for transition model $\eta_\beta$, initial parameters $\theta_0 = (\alpha_0, \beta_0)$, expert demonstrations $\{s_{0:H}\}$, context length $L$, batch size $m$, number of prior sampling steps $K$ and step sizes $s$.
Output: $\theta_N = (\alpha_N, \beta_N)$.
Reorganize $\{s_{0:H}\}$ to to state sequenec segments $(s_{t-L+1}, \cdots, s_{t+1})$ with length $L + 1$.
Use energy-based policy with $\alpha_0$ collect transitions to fill in the replay buffer.
Use transitions in replay buffer to pre-train transition model $\beta_0$.
for $t = 0$ to $N - 1$ do
    Demo sampling Sample observed examples $(s_{t-L+1}, \cdots, s_{t+1})_{i=1}^{m}$.
    Prior sampling: Sample $\{\hat{a}_t\}_{i=1}^{m}$ using Eq. (15) with $K_0$ iterations and stepsize $s_0$.
    Policy learning: Update $\alpha_t$ to $\alpha_{t+1}$ by Eq. (19) with learning rate $\eta_\alpha$.
    Transition learning: Update replay buffer with trajectories from current policy model $\alpha_{t+1}$, then update $\beta_t$ to $\beta_{t+1}$ by Eq. (9) with learning rate $\eta_\beta$.
end for
```

B A Decision-making Problem in MLE

Let the ground-truth distribution of demonstrations be $p^*(s_{0:T})$, and the learned marginal distributions of state sequences be $p_\theta(s_{0:T})$. Eq. (5) is an empirical estimate of

$$\mathbb{E}_{p^*(s_{0:T})}[\log p_\theta(s_{0:T})] = \mathbb{E}_{p^*(s_0)}\left[\log p^*(s_0) + \mathbb{E}_{p^*(s_{1:T}|s_0)}[\log p_\theta(s_{1:T}|s_0)]\right]. \tag{20}$$

We can show that a sequential decision-making problem can be constructed to maximize the same objective. To start off, suppose the MLE yields the maximum, we will have $p_{\theta *} = p^*$.

Define $V^*(s_0) := \mathbb{E}_{p^*(s_{1:T}|s_0)}[\log p^*(s_{1:T}|s_0)]$, we can generalize it to have a V *function*

$$V^*(s_{0:t}) := \mathbb{E}_{p^*(s_{t+1:T}|s_{0:t})}[\log p^*(s_{t+1:T}|s_{0:t})], \tag{21}$$

which comes with a Bellman optimality equation:

$$V^*(s_{0:t}) = \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r(s_{t+1}, s_{0:t}) + V^*(s_{0:t+1})\right], \tag{22}$$

with $r(s_{t+1}, s_{0:t}) := \log p^*(s_{t+1}|s_{0:t}) = \log \int p_{\alpha *}(a_t|s_{0:t}) p_{\beta *}(s_{t+1}|s_t, a_t) da_t$, $V^*(s_{0:T}) := 0$. It is worth noting that the r defined above involves the optimal policy, which may not be known a priori. We can resolve this by replacing it with r_α for an arbitrary policy $p_\alpha(a_t|s_{0:t})$. All Bellman identities and updates should still hold. Anyways, involving the current policy in the reward function should not appear to be too odd given the popularity of maximum entropy RL [20, 24].

The entailed Bellman update, *value iteration*, for arbitrary V and α is

$$V(s_{0:t}) = \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r_\alpha(s_{0:t}, s_{t+1}) + V(s_{0:t+1})\right]. \tag{23}$$

We then define $r(s_{t+1}, a_t, s_{0:t}) := r(s_{t+1}, s_{0:t}) + \log p_{\alpha *}(a_t|s_{0:t})$ to construct a Q function:

$$Q^*(a_t; s_{0:t}) := \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r(s_{t+1}, a_t, s_{0:t}) + V^*(s_{0:t+1})\right], \tag{24}$$

which entails a Bellman update, *Q backup*, for arbitrary α, Q and V

$$Q(a_t; s_{0:t}) = \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r_\alpha(s_{0:t}, a_t, s_{t+1}) + V(s_{0:t+1})\right]. \tag{25}$$

Also note that the V and Q in identities Eq. (23) and Eq. (25) respectively are not necessarily associated with the policy $p_\alpha(a_t|s_{0:t})$. Slightly overloading the notations, we use Q^α, V^α to denote the expected returns from policy $p_\alpha(a_t|s_{0:t})$.

By now, we finish the construction of atomic algebraic components and move on to check if the relations between them align with the algebraic structure of a sequential decision-making problem [9].

We first prove the construction above is valid at optimality.

Lemma 1. *When $f_\alpha(a_t; s_{0:t}) = Q^*(a_t; s_{0:t}) - V^*(s_{0:t})$, $p_\alpha(a_t|s_{0:t})$ is the optimal policy.*

Proof. Note that the construction gives us

$$\begin{aligned} Q^*(a_t; s_{0:t}) &= \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r(s_{t+1}, s_{0:t}) + \log p_{\alpha *}(a_t|s_{0:t}) + V^*(s_{0:t+1})\right] \\ &= \log p_{\alpha *}(a_t|s_{0:t}) + \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r(s_{t+1}, s_{0:t}) + V^*(s_{0:t+1})\right] \\ &= \log p_{\alpha *}(a_t|s_{0:t}) + V^*(s_{0:t}). \end{aligned} \tag{26}$$

Obviously, $Q^*(a_t; s_{0:t})$ lies in the hypothesis space of $f_\alpha(a_t; s_{0:t})$. □

Lemma 1 indicates that we need to either parametrize $f_\alpha(a_t; s_{0:t})$ or $Q(a_t; s_{0:t})$.

While Q^α and V^α are constructed from the optimality, the derived Q^α and V^α measure the performance of an interactive agent when it executes with the policy $p_\alpha(a_t|s_{0:t})$. They should be consistent with each other.

Lemma 2. *$V^\alpha(s_{0:t})$ and $\mathbb{E}_{p_\alpha(a_t|s_{0:t})}[Q^\alpha(a_t; s_{0:t})]$ yield the same optimal policy $p_{\alpha *}(a_t|s_{0:t})$.*

Proof.

$$\begin{aligned} &\mathbb{E}_{p_\alpha(a_t|s_{0:t})}[Q^\alpha(a_t; s_{0:t})] := \mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r(s_{t+1}, a_t, s_{0:t}) + V^\alpha(s_{0:t+1})\right]\right] \\ =&\mathbb{E}_{p_\alpha(a_t|s_{0:t})}\left[\mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[\log p_\alpha(a_t|s_{0:t}) + r(s_{t+1}, s_{0:t}) + V^\alpha(s_{0:t+1})\right]\right] \\ =&\mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r(s_{t+1}, s_{0:t}) - \mathcal{H}_\alpha(a_t|s_{0:t}) + V^\alpha(s_{0:t+1})\right] \\ =&V^\alpha(s_{0:t}) - \mathcal{H}_\alpha(a_t|s_{0:t}) - \sum_{k=t+1}^{T-1} \mathbb{E}_{p^*(s_{t+1:k}|s_{0:t})}[\mathcal{H}_\alpha(a_k|s_{0:k})], \end{aligned} \tag{27}$$

where the last line is derived by recursively applying the Bellman equation in the line above until $s_{0:T}$ and then applying backup with Eq. (23). As an energy-based policy, $p_\alpha(a_t|s_{0:t})$'s entropy is inherently maximized [66]. Therefore, within the hypothesis space, $p_{\alpha^*}(a_t|s_{0:t})$ that optimizes $V^\alpha(s_{0:t})$ also leads to optimal expected return $\mathbb{E}_{p_\alpha(a_t|s_{0:t})}[Q^\alpha(a_t;s_{0:t})]$. □

If we parametrize the policy as $p_\alpha(a_t|s_{0:t}) \propto \exp(Q^\alpha(a_t;s_{0:t}))$, the logarithmic normalizing constant $\log Z^{\alpha_k}(s_{0:t})$ will be the *soft V function* in maximum entropy RL [21–23]

$$V^\alpha_{soft}(s_{0:t}) := \log \int \exp(Q^\alpha(a_t;s_{0:t}))da_t, \tag{28}$$

even if the reward function is defined differently. We can further show that Bellman identities and backup updates above can entail RL algorithms that achieve optimality of the decision-making objective V^α, including *soft policy iteration* [20]

$$p_{\alpha_{k+1}}(a_t|s_{0:t}) \leftarrow \frac{\exp(Q^{\alpha_k}(a_t;s_{0:t}))}{Z^{\alpha_k}(s_{0:t})}, \forall s_{0:t}, k \in [0, 1, ...M]; \tag{29}$$

and *soft Q iteration* [21]

$$\begin{aligned} Q^{\alpha_{k+1}}(a_t;s_{0:t}) &\leftarrow \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r_\alpha(s_{0:t},a_t,s_{t+1}) + V^{\alpha_k}_{soft}(s_{0:t+1})\right], \forall s_{0:t}, a_t, \\ V^{\alpha_{k+1}}_{soft}(s_{0:t}) &\leftarrow \log \int \exp(Q^{\alpha_k}(a;s_{0:t}))da, \forall s_{0:t}, k \in [0, 1, ...M]. \end{aligned} \tag{30}$$

Lemma 3. *If $p^*(s_{t+1}|s_{0:t})$ is accessible and $p_{\beta^*}(s_{t+1}|s_t,a_t)$ is known, soft policy iteration and soft Q learning both converge to $p_{\alpha^*}(a_t|s_{0:t}) = p_{Q^*}(a_t|s_{0:t}) \propto \exp(Q^*(a_t;s_{0:t}))$ under certain conditions.*

Proof. See the convergence proof by Ziebart [20] for *soft policy iteration* and the proof by Fox et al. [21] for *soft Q learning*. The latter requires Markovian assumption. But under some conditions, it can be extended to non-Markovian domains in the same way as proposed by Majeed and Hutter [67]. □

Lemma 3 means given $p^*(s_{t+1}|s_{0:t})$ and $p_{\beta^*}(s_{t+1}|s_t,a_t)$, we can recover p_{α^*} through reinforcement learning methods, instead of the proposed MLE. So $p_\alpha(a_t|s_{0:t})$ is a viable policy space for the constructed sequential decision-making problem.

Together, Lemma 1, Lemma 2 and Lemma 3 provide constructive proof for a valid sequential decision-making problem that maximizes the same objective of MLE, described by Theorem 1.

Theorem 1. *Assuming the Markovian transition $p_{\beta^*}(s_{t+1}|s_t,a_t)$ is known, the ground-truth conditional state distribution $p^*(s_{t+1}|s_{0:t})$ for demonstration sequences is accessible, we can construct a sequential decision-making problem, based on a reward function $r_\alpha(s_{t+1},s_{0:t}) := \log \int p_\alpha(a_t|s_{0:t})p_{\beta^*}(s_{t+1}|s_t,a_t)da_t$ for an arbitrary energy-based policy $p_\alpha(a_t|s_{0:t})$. Its objective is*

$$\sum_{t=0}^{T} \mathbb{E}_{p^*(s_{0:t})}[V^{p_\alpha}(s_{0:t})] = \mathbb{E}_{p^*(s_{0:T})}\left[\sum_{t=0}^{T}\sum_{k=t}^{T} r_\alpha(s_{k+1};s_{0:k})\right],$$

where $V^{p_\alpha}(s_{0:t}) := E_{p^*(s_{t+1:T}|s_{0:t})}[\sum_{k=t}^{T} r_\alpha(s_{k+1};s_{0:k})]$ is the value function for p_α. This objective yields the same optimal policy as the Maximum Likelihood Estimation $\mathbb{E}_{p^*(s_{0:T})}[\log p_\theta(s_{0:T})]$.

If we further define a reward function $r_\alpha(s_{t+1},a_t,s_{0:t}) := r_\alpha(s_{t+1},s_{0:t}) + \log p_\alpha(a_t|s_{0:t})$ to construct a Q function for p_α

$$Q^{p_\alpha}(a_t;s_{0:t}) := \mathbb{E}_{p^*(s_{t+1}|s_{0:t})}\left[r_\alpha(s_{t+1},a_t,s_{0:t}) + V^{p_\alpha}(s_{0:t+1})\right].$$

The expected return of $Q^{p_\alpha}(a_t;s_{0:t})$ forms an alternative objective

$$\mathbb{E}_{p_\alpha(a_t|s_{0:t})}[Q^{p_\alpha}(a_t;s_{0:t})] = V^{p_\alpha}(s_{0:t}) - \mathcal{H}_\alpha(a_t|s_{0:t}) - \sum_{k=t+1}^{T-1} \mathbb{E}_{p^*(s_{t+1:k}|s_{0:t})}[\mathcal{H}_\alpha(a_k|s_{0:k})]$$

that yields the same optimal policy, for which the optimal $Q^*(a_t;s_{0:t})$ can be the energy function.

Only under certain conditions, this sequential decision-making problem is solvable through non-Markovian extensions of the maximum entropy reinforcement learning algorithms.

C More results on Curve Planning

The energy function is parameterized by a small MLP with one hidden layer and $4 * L$ hidden neurons, where L is the context length. In short-run Langevin dynamics, the number of samples, the number of sampling steps, and the stepsize are 4, 20 and 1 respectively. We use Adam optimizer with a learning rate 1e-4 and batch size 64. Here we present the complete result in Fig. A1 with different training steps under context length 1 2 4 6, the acceptance rate and residual error of the testing trajectories, as well as the behavior cloning results. We can see that even with sufficient context, BC performs worse than LanMDP. Also, from the result of context length 6 we can see that excessive expressivity does not impair performance, it just requires more training.

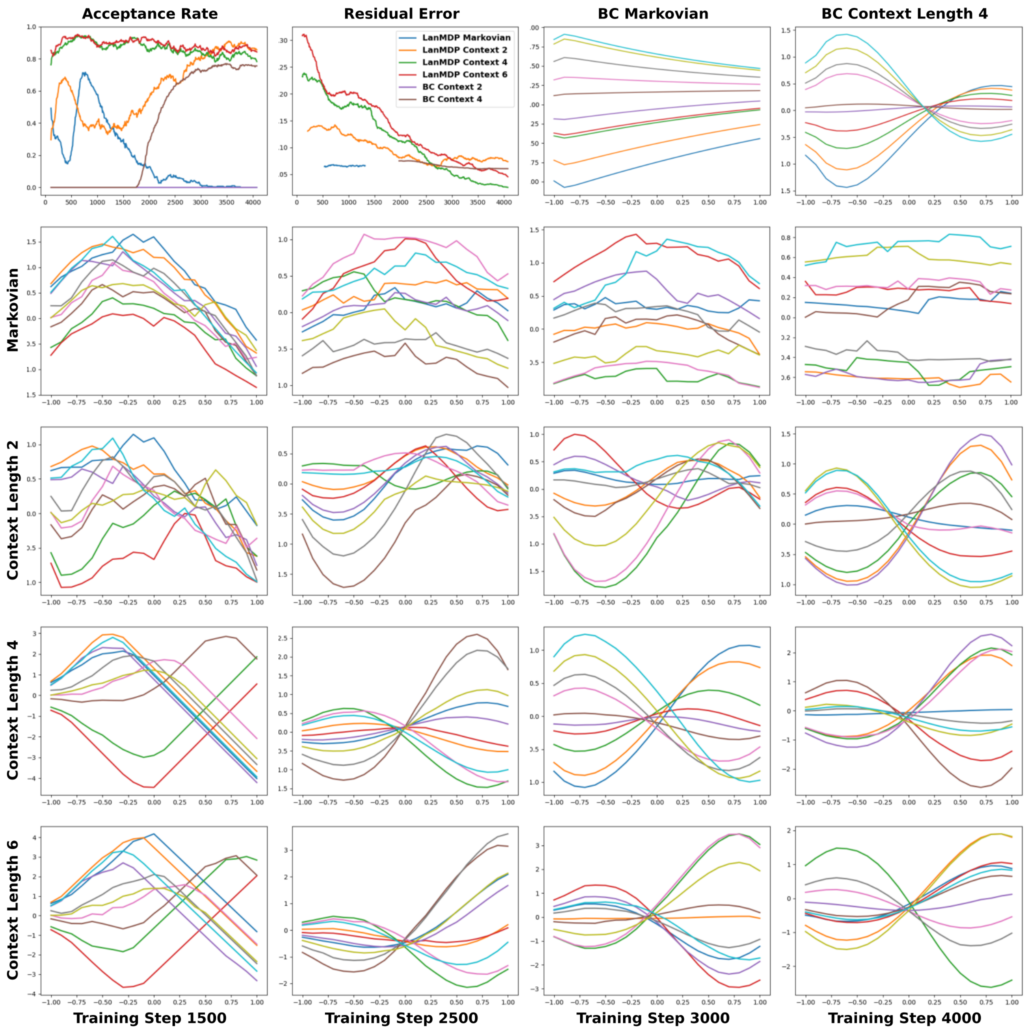

Figure A1: More results for cubic curve generation

D Implementation Details of MuJoCo Environment

This section delineates the configurations for the MuJoCo environments utilized in our research. In particular, we employ standard environment horizons of 500 and 50 for Cartpole-v1 and Reacher-v2, respectively. Meanwhile, for Swimmer-v2, Hopper-v2, and Walker2d-v2, we operate within an environment horizon set at 400 as referenced in previous literature [52, 68–72]. Additional specifications are made for Hopper-v2 and Walker2d-v2, where the velocity of the center of mass was integrated into the state parameterization [52, 68, 70, 72]. We leverage PPO [45] approach to train the expert policy until it reaches (approximately) 450, -10, 40, 3000, 2000 for Cartpole-v1, Reacher-v2, Swimmer-v2, Hopper-v2, Walker2d-v2 respectively. It should be noted that all results disclosed in the experimental section represent averages over five random seeds. Comparative benchmarks include BC [46], BCO [37], GAIL [35], and GAIFO [7]. MobILE [52] is a recent method for Markovian model-based imitation from observation. However, we failed to reproduce the expected performance utilizing various sets of demonstrations, so it is prudently omitted from the present displayed result. We specifically point out that BC/GAIL algorithms are privy to expert actions, however, our algorithm is not. We report the mean of the best performance achieved by BC/BCO with five random seeds, even though these peak performances may transpire at varying epochs. For BC, we executed the supervised learning algorithm for 200 iterations. The BCO/GAIL algorithms are run with an equivalent number of online samples as LanMDP for a fair comparison. All benchmarking is performed using a single 3090Ti GPU and implemented using the PyTorch framework. Notably, in our codebase, the modified environments of Hopper-v2 and Walker2d-v2 utilize MobILE's implementation [52]. Referring to the results in the main text, our presentation of normalized results in bar graph form is derived by normalizing each algorithm's performance (mean/standard deviation) against the expert mean. For Reacher-v2, due to the inherently negative rewards, we first add a constant offset of 20 to each algorithm's performance, thus converting all values to positive before normalizing them against the mean of expert policy.

We parameterize both the policy model and the transition model as MLPs, and the non-linear activation function is Swish and LeakyReLU respectively. We use AdamW to optimize both policy and transition. To stabilize training, we prefer using actions around which the transition model is more certain for computing the expectation over importance-weighted prior distribution in Eq. (19). Therefore, we use a model ensemble with two transition models and use the disagreement between these two models to measure the uncertainty of the sampled actions. We implement Algorithm 2 for all experiments to avoid expensive computation of the gradient for the transition model in posterior sampling. As for better and more effective short-run Langevin sampling, we use a polynomially decaying schedule for the step size as recommended in [73]. We also use weakly L2 regularized energy magnitudes and clip gradient steps like [74], choosing to clip the total amount of change value, i.e. after the gradient and noise have been combined. To realize more delicate decision-making, another trick in Implicit Behavior Clone [75] is also adopted for the inference/testing stage that we continue running the MCMC chain after the step size reaches the smallest in the polynomial schedule until we get twice as many inference Langevin steps as were used during training.

Hyper-parameters are listed in Table 3. Other hyperparameters that are not mentioned here are left as default in PyTorch. Also, note that the Cartpole-v1 task has no parameters for sampling because expectation can be calculated analytically.

Table 3: Hyper-parameter list of MuJoCo experiments

Parameter	Cartpole-v1	Reacher-v2	Swimmer-v2	Hopper-v2	Walker2d-v2
Environment Specification					
Horizon	500	50	400	400	400
Expert Performance ($\approx$)	450	-10	40	3000	2000
Transition Model					
Architecture(hidden;layers)	MLP(64;4)	MLP(64;4)	MLP(128;4)	MLP(512;4)	MLP(512;4)
Optimizer(LR)	3e-3	3e-3	3e-3	3e-3	3e-3
Batch Size	2500	20000	20000	32768	32768
Replay Buffer Size	2500	20000	20000	200000	200000
Policy Model (with context length L)					
Architecture(hidden;layers)	MLP(150*L;4)	MLP(150*L;4)	MLP(150*L;4)	MLP(512*L;4)	MLP(512*L;4)
Learning rate	1e-3	1e-2	1e-2	1e-2	5e-3
Batch Size	2500	20000	20000	32768	32768
Number of test trajectories	5	20	20	50	50
Sampling Parameters					
Number of prior samples	\	8	8	8	8
Number of Langevin steps	\	100	100	100	100
Langevin initial stepsize	\	10	10	10	10
Langevin ending stepsize	\	1	1	1	1