Supplementary of Learning Energy-Based Prior Model with Diffusion-Amortized MCMC

Contents

Preprint. Under review.

A Related Work

Energy-based prior model Energy-Based Models (EBMs) [1–5] play an important role in generative modeling. Pang et al. [6] propose to learn an EBM as a prior model in the latent space of deep latent variable models; it greatly improves the model expressivity over those with non-informative priors and brings strong performance on downstream tasks, *e.g.*, image segmentation, text modeling, molecule generation, and trajectory prediction [7–10]. However, learning both EBMs and latent space EBMs require MCMC sampling to estimate the learning gradients, which requires a large amount of iterations to converge when the target distributions are high-dimensional or highly multi-modal. Typical choices of sampling with non-convergent short-run MCMC [2] in practice can lead to poor generation quality, malformed energy landscapes [2, 8, 11], biased estimation of the model parameter and instability in training [3–5, 11]. In this work, we consider learning valid amortization of the potentially long-run MCMC for learning energy-based priors; the proposed model shows reliable sampling quality in practice.

Denoising diffusion probabilistic model Denoising Diffusion Probabilistic Models (DDPMs) [12–15], originating from Sohl-Dickstein et al. [12], learn the generative process by recovering the observed data from a sequence of noise-perturbed versions of the data. The learning objective can be viewed as a variant of the denoising score matching objective [16]. As pointed out in [12, 13], the sampling procedure of DDPM with ϵ-prediction parametrization resembles Langevin Dynamics (LD) of an EBM; ϵ (predicted noise) plays a similar role to the gradient of the log density [13]. To be specific, learning a DDPM with ϵ-prediction parameterization is equivalent to fitting the finite-time marginal of a sampling chain resembling annealed Langevin dynamics [13–15]. Inspired by this connection, we propose to amortize the long-run MCMC in learning energy-based prior by iteratively distilling the short-run sampling chain segments with a diffusion-based sampler. We provide empirical and theoretical evidence that the resulting sampler is a valid long-run chain sampler.

Amortized MCMC The amortized MCMC technique is formally brought up by Li et al. [17], which incorporates feedback from MCMC back to the parameters of the amortizer distribution q_ϕ. It is concurrently and independently proposed by Xie et al. [18] as the MCMC teaching framework. Methods under this umbrella term [17–22] generally learns the amortizer by minimizing the divergence (typically the Kullbeck-Leibler Divergence (KLD)) between the improved distribution and its initialization, i.e., $\mathcal{D}[\mathcal{K}_T q_{\phi_{k-1}} || q_\phi]$, where $\mathcal{K}_T$ represents T-step MCMC transition kernel and $q_{\phi_{k-1}}$ represents the current amortizer. The diffusion-based amortization proposed in this work can be viewed as an instantiation of this framework, while our focus is on learning the energy-based prior. Compared with previous methods, our method i) specifically exploits the connection between EBMs and DDPMs and is suitable for amortizing the prior and posterior sampling MCMC in learning energy-based prior, and ii) resides in the lower-dimensional latent space and enables faster sampling and better convergence.

More methods for learning EBM Several techniques other than short-run MCMC have been proposed to learn the EBM. In the seminal work, Hinton [23] proposes to initialize Markov chains using real data and run several steps of MCMC to obtain samples from the model distribution. Tieleman [24] proposes to start Markov chains from past samples in the previous sampling iteration, known as Persistent Contrastive Divergence (PCD) or persistent chain sampling, to mimic the long-run sampling chain. Nijkamp et al. [4] provide comprehensive discussions about tuning choices for LD such as the step size s and sampling steps T to obtain stable long-run samples for persistent training. [3, 5, 25] employ a hybrid of persistent chain sampling and short-run sampling by maintaining a buffer of previous samples. The methods draw from the buffer or initialize the short-run chain with noise distribution with some pre-specified probability. Another branch of work, stemmed from [26], considers discriminative contrastive estimation to avoid MCMC sampling. Gao et al. [27] use a normalizing flow [28] as the base distribution for contrastive estimation. Aneja et al. [29] propose to estimate the energy-based prior model based on the prior of a pre-trained VAE [30] by noise contrastive estimation. More recently, Xiao and Han [31] learn a sequence of EBMs in the latent space with adaptive multi-stage NCE to further improve the expressive power of the model.

B Theoretical Discussion

B.1 Monotonically Decreasing KLD

We state in the main text that $\mathcal{D}[q_{\boldsymbol{\phi}_k}||\pi] \leqslant \mathcal{D}[q_{\boldsymbol{\phi}_{k-1}}||\pi]$, where π is the stationary distribution. To show this, we first provide a proof of $\mathcal{D}[\pi_{t+T}||\pi] \leqslant \mathcal{D}[\pi_t||\pi]$, where π_t and π_{t+T} are the distributions of $\boldsymbol{z}$ at t-th and $(t+T)$-th iteration, respectively. This is a known result from [32], and we include it here for completeness.

$$\begin{aligned}\mathcal{D}[\pi_t||\pi] &= \mathbb{E}_{\pi_t(\boldsymbol{z}_t)}\left[\log\frac{\pi_t(\boldsymbol{z}_t)}{\pi(\boldsymbol{z}_t)}\right] = \mathbb{E}_{\pi_t(\boldsymbol{z}_t),\mathcal{K}(\boldsymbol{z}_{t+1}|z_t)}\left[\log\frac{\pi_t(\boldsymbol{z}_t)\mathcal{K}(\boldsymbol{z}_{t+1}|z_t)}{\pi(\boldsymbol{z}_t)\mathcal{K}(\boldsymbol{z}_{t+1}|z_t)}\right]\\ &\overset{(i)}{=} \mathbb{E}_{\pi_{t+1}(\boldsymbol{z}_{t+1}),\mathcal{K}'_{\pi_{t+1}}(\boldsymbol{z}_t|z_{t+1})}\left[\log\frac{\mathcal{K}'_{\pi_{t+1}}(\boldsymbol{z}_t|z_{t+1})\pi_{t+1}(\boldsymbol{z}_{t+1})}{\mathcal{K}'_{\pi}(\boldsymbol{z}_t|z_{t+1})\pi(\boldsymbol{z}_{t+1})}\right]\\ &= \mathcal{D}[\pi_{t+1}||\pi] + \mathbb{E}_{\pi_{t+1}(\boldsymbol{z}_{t+1})}\mathcal{D}[\mathcal{K}'_{\pi_{t+1}}(\boldsymbol{z}_t|z_{t+1})||\mathcal{K}'_{\pi}(\boldsymbol{z}_t|z_{t+1})]\\ &\overset{(ii)}{\geqslant} \mathcal{D}[\pi_{t+1}||\pi],\end{aligned} \tag{1}$$

where we denote the forward-time transition kernel as $\mathcal{K}$ and the reverse-time kernel as $\mathcal{K}'$. (i) holds because we are just re-factorizing the joint density of $[\boldsymbol{z}_t, \boldsymbol{z}_{t+1}]$: $\pi_t(\boldsymbol{z}_t)\mathcal{K}(\boldsymbol{z}_{t+1}|z_t) = \mathcal{K}'_{\pi_{t+1}}(\boldsymbol{z}_t|z_{t+1})\pi_{t+1}(\boldsymbol{z}_{t+1})$ and $\pi(\boldsymbol{z}_t)\mathcal{K}(\boldsymbol{z}_{t+1}|z_t) = \mathcal{K}'_{\pi}(\boldsymbol{z}_t|z_{t+1})\pi(\boldsymbol{z}_{t+1})$. (ii) holds because the KLD is non-negative. We can see that $\mathcal{D}[\pi_{t+T}||\pi] \leqslant \mathcal{D}[\pi_t||\pi]$ is a direct result from Eq. (1), and that $\pi_t \to \pi$ as $t \to \infty$ under proper conditions [33].

In the main text, we describe the update rule of the sampler $q_{\boldsymbol{\phi}}$ as follows:

$$q_{\boldsymbol{\phi}_k} \leftarrow \underset{q_{\boldsymbol{\phi}}\in\mathcal{Q}}{\arg\min}\,\mathcal{D}[q_{\boldsymbol{\phi}_{k-1},T}||q_{\boldsymbol{\phi}}],\ q_{\boldsymbol{\phi}_{k-1},T} := \mathcal{K}_T q_{\boldsymbol{\phi}_{k-1}},\ q_{\boldsymbol{\phi}_0} \approx \pi_0. \tag{2}$$

In the ideal case, we can assume that the objective in Eq. (2) is properly optimized and that $\{q_{\boldsymbol{\phi}}\}$ is expressive enough to parameterize each $q_{\boldsymbol{\phi}_{k-1},T}$. With $q_{\boldsymbol{\phi}_0} \approx \pi_0$ and $q_{\boldsymbol{\phi}_k} \approx \mathcal{K}_T q_{\boldsymbol{\phi}_{k-1}}$, we can conclude that $\mathcal{D}[q_{\boldsymbol{\phi}_k}||\pi] \leqslant \mathcal{D}[q_{\boldsymbol{\phi}_{k-1}}||\pi]$ for each $k = 1, ..., K$ according to Eq. (1). We will discuss in the following section the scenario where we apply gradient-based methods to minimize $\mathcal{D}[q_{\boldsymbol{\phi}_{k-1},T}||q_{\boldsymbol{\phi}}]$ and approximate Eq. (2) for the update from $q_{\boldsymbol{\phi}_{k-1}}$ to $q_{\boldsymbol{\phi}_k}$.

B.2 Discussion about Diffusion-Based Amortization

We can see that the statement in Appendix B.1 holds when $q_{\boldsymbol{\phi}_k}$ is a close approximation of $q_T := \mathcal{K}_T q_{\boldsymbol{\phi}_{k-1}}$. This motivates our choice of employing DDPM to amortize the LD transition, considering its capability of close approximation to the given distribution $\{q_T\}$. Based on the derivation of DDPM learning objective in [34], we know that

$$\begin{aligned}\underset{q_{\boldsymbol{\phi}}}{\arg\min}\,\mathcal{D}[q_T||q_{\boldsymbol{\phi}}] &= \underset{q_{\boldsymbol{\phi}}}{\arg\min} -\mathcal{H}(q_T) + \mathcal{H}(q_T, q_{\boldsymbol{\phi}}) = \underset{q_{\boldsymbol{\phi}}}{\arg\min} -\mathbb{E}_{q_T}\left[\log q_{\boldsymbol{\phi}}\right]\\ &\leqslant \underset{q_{\boldsymbol{\phi}}}{\arg\min}\,\mathbb{E}_{\boldsymbol{\epsilon},\lambda}\left[\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\boldsymbol{z}_\lambda) - \boldsymbol{\epsilon}\|_2^2\right] \approx \underset{q_{\boldsymbol{\phi}}}{\arg\min}\,\frac{1}{N}\sum_{j=1}^N\left[\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\boldsymbol{z}_{j,\lambda_j}) - \boldsymbol{\epsilon}_j\|_2^2\right],\end{aligned} \tag{3}$$

where $\boldsymbol{z}_\lambda$ is draw from $q(\boldsymbol{z}_\lambda|\boldsymbol{z}_0) = \mathcal{N}(\boldsymbol{z}_\lambda; \beta_\lambda\boldsymbol{z}_0, \sigma_\lambda^2\mathbf{I}_d)$. $\boldsymbol{z}_0 \sim q_T$. $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ and λ is drawn from a distribution of log noise-to-signal ratio $p(\lambda)$. In practice, we use Monte-Carlo average to approximate the objective and employ a gradient-based update rule for $q_{\boldsymbol{\phi}}$:

$$\boldsymbol{\phi}_{k-1}^{(i+1)} \leftarrow \boldsymbol{\phi}_{k-1}^{(i)} - \eta\nabla_{\boldsymbol{\phi}}\frac{1}{N}\sum_{j=1}^N\left[\|\boldsymbol{\epsilon}_{\boldsymbol{\phi}}(\boldsymbol{z}_{j,\lambda_j})\boldsymbol{\epsilon}_j\|_2^2\right],\ \boldsymbol{\phi}_k^{(0)} \leftarrow \boldsymbol{\phi}_{k-1}^{(M)},\ i = 0, 1, ..., M-1. \tag{4}$$

This can be viewed as a M-estimation of $\boldsymbol{\phi}$. Recall that $q_T = \mathcal{K}_T q_{\boldsymbol{\phi}_{k-1}}$, we can construct $\boldsymbol{\phi}_k = \left(\boldsymbol{\phi}_{k-1}, \tilde{\boldsymbol{\phi}}\right)$ to minimize the KLD, where $\tilde{\boldsymbol{\phi}}$ models the transition kernel $\mathcal{K}_T$. Therefore, initializing $q_{\boldsymbol{\phi}}$ to be optimized with $q_{\boldsymbol{\phi}_{k-1}}$, we are effectively maximizing $\mathcal{L}(\tilde{\boldsymbol{\phi}}) = \frac{1}{N}\sum_{j=1}^N \log p_{\tilde{\boldsymbol{\phi}}}(\hat{\boldsymbol{z}}_j|\boldsymbol{z}_j)$, where $\{\boldsymbol{z}_j\}$ are from $q_{\boldsymbol{\phi}_{k-1}}$ and $\{\hat{\boldsymbol{z}}_j\}$ are from $\mathcal{K}_T q_{\boldsymbol{\phi}_{k-1}}$. Let $\tilde{\boldsymbol{\phi}}$ be the result of M-estimation and $\tilde{\boldsymbol{\phi}}^*$ be the true target parameter. Then based on the derivation in [11], asymptotically we have

$$\sqrt{N}\left(\tilde{\boldsymbol{\phi}} - \tilde{\boldsymbol{\phi}}^*\right) \to \mathcal{N}\left(0, \mathcal{I}\left(\tilde{\boldsymbol{\phi}}^*\right)^{-1}\right), \tag{5}$$

where $\mathcal{I}\left(\tilde{\boldsymbol{\phi}}^*\right) = \mathbb{E}_{\hat{\boldsymbol{z}},\boldsymbol{z}}\left[-\nabla^2 \log p_{\tilde{\boldsymbol{\phi}}}(\hat{\boldsymbol{z}}|\boldsymbol{z})\right]$ is the Fisher information matrix. This interpretation tells us that i) when the sample size N is large, the estimation $\tilde{\boldsymbol{\phi}}$ is asymptotically unbiased, and ii) if we want to obtain the estimation $\tilde{\boldsymbol{\phi}}$ with a few gradient-based updates, then the eigenvalues of the Fisher information matrix would be relatively small but non-zero. ii) suggests that $\mathcal{K}_T q_{\boldsymbol{\phi}_{k-1}}$ should be significantly different from $q_{\boldsymbol{\phi}_{k-1}}$, which is confirmed by [19] and our preliminary experiments, but it should not be too far away because that would require more gradient-based updates. We find that setting $T = 30$ and $M = 6$ works well in the experiments.

B.3 Further Discussion about the Learning Algorithm

For completeness, we first derive the learning gradients for updating $\boldsymbol{\theta}$.

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \frac{1}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\nabla_{\boldsymbol{\theta}} \int_{\boldsymbol{z}} p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})d\boldsymbol{z} = \int_{\boldsymbol{z}} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})d\boldsymbol{z} \\
&= \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})\right] \\
&= \left(\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\alpha}} \log p_{\boldsymbol{\alpha}}(\boldsymbol{z})\right], \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\beta}} \log p_{\boldsymbol{\beta}}(\boldsymbol{x}|\boldsymbol{z})\right]\right) \\
&= (\underbrace{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\alpha}} f_{\boldsymbol{\alpha}}(\boldsymbol{z})\right] - \mathbb{E}_{p_{\boldsymbol{\alpha}}(\boldsymbol{z})}\left[\nabla_{\boldsymbol{\alpha}} f_{\boldsymbol{\alpha}}(\boldsymbol{z})\right]}_{\delta_{\boldsymbol{\alpha}}(\boldsymbol{x})}, \underbrace{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\beta}} \log p_{\boldsymbol{\beta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\delta_{\boldsymbol{\beta}}(\boldsymbol{x})}).
\end{aligned} \tag{6}$$

We can see that $\boldsymbol{\theta}$ is estimated in a MLE-by-EM style. The learning gradient is the same as that of directly maximizing the observed data likelihood, while we need to approximate the expectations in Eq. (6). Estimating the expectations is like the E-step, and update $\boldsymbol{\theta}$ with Eq. (6) is like the M-step in the EM algorithm. The proposed diffusion-based amortization brings better estimation of the expectations in the E-step, and incorporate the feedback from the M-step by running prior and posterior sampling LD as follows

$$\begin{aligned}
\boldsymbol{z}_{t+1} &= \boldsymbol{z}_t + \frac{s^2}{2}\nabla_{\boldsymbol{z}_t} \underbrace{\left(f_{\boldsymbol{\alpha}}(\boldsymbol{z}_t) - \frac{1}{2}\|\boldsymbol{z}_t\|_2^2\right)}_{\log p_{\boldsymbol{\alpha}}(\boldsymbol{z}_t)} + s\boldsymbol{w}_t, \\
\boldsymbol{z}_{t+1} &= \boldsymbol{z}_t + \frac{s^2}{2}\nabla_{\boldsymbol{z}_t} \underbrace{\left(-\frac{\|\boldsymbol{x} - g_{\boldsymbol{\beta}}(\boldsymbol{z}_t)\|_2^2}{2\sigma^2} + f_{\boldsymbol{\alpha}}(\boldsymbol{z}_t) - \frac{1}{2}\|\boldsymbol{z}_t\|_2^2\right)}_{\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) + C} + s\boldsymbol{w}_t,
\end{aligned} \tag{7}$$

to obtain training data. Here $t = 0, 1, ..., T$. $\boldsymbol{z}_0 \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ for posterior sampling and $\boldsymbol{z}_0 \sim q_{\boldsymbol{\phi}}(\boldsymbol{z})$ for prior sampling. $\boldsymbol{w}_t \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_d)$. Note that we plug-in $p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})$ for the target distribution of posterior sampling LD. This is because given the observed data $\boldsymbol{x}$, by Bayes' rule we know that $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \propto p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) = p_{\boldsymbol{\beta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\alpha}}(\boldsymbol{z})$. The whole learning iteration can be viewed as a variant of the variational EM algorithm [35].

C Network Architecture and Training Details

Architecture The energy score network f_α uses a simple fully-connected structure throughout the experiments. We describe the architecture in details in Table 1. The generator network has a simple deconvolution structure similar to DCGAN [36] shown in Table 2. The denoising diffusion model is implemented by a light-weight MLP-based U-Net [37] structure (Table 3). The encoder network to embed the observed images has a fully convolutional structure [38], as shown in Table 4.

Table 1: **Network structures of the energy score network.** LReLU denotes the Leaky ReLU activation function. The slope in Leaky ReLU is set to 0.2. For the SVHN and CelebA datasets, we use `nz=100`. For the CIFAR-10 and CelebA-HQ datasets, we use `nz=128`. We use `nz=8` for anomaly detection on the MNIST dataset, and `nz=7168` for GAN inversion. We use `ndf=512` for GAN inversion and `ndf=200` for the rest experiments.

Layers	Out Size
Input: $\boldsymbol{z}$	nz $\in \{8, 100, 128, 7168\}$
Linear, LReLU	ndf $\in \{200, 512\}$
Linear, LReLU	ndf $\in \{200, 512\}$
Linear	1

Table 2: **Network structures of the generator networks** used for the SVHN, CelebA, CIFAR-10, CelebA-HQ and MNIST (from top to bottom) datasets. For GAN inversion, we use the StyleGAN [39] structure as our generator network. ConvT(n) indicates a transposed convolutional operation with n output channels. We use `ngf=64` for the SVHN dataset and `ngf=128` for the rest. LReLU indicates the Leaky-ReLU activation function. The slope in Leaky ReLU is set to be 0.2.

Layers	Out Size	Stride
Input: $\boldsymbol{z}$	1x1x100	-
4x4 ConvT(ngf x 8), LReLU	4x4x(ngf x 8)	1
4x4 ConvT(ngf x 4), LReLU	8x8x(ngf x 4)	2
4x4 ConvT(ngf x 2), LReLU	16x16x(ngf x 2)	2
4x4 ConvT(3), Tanh	32x32x3	2
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{z}$	1x1x100	-
4x4 ConvT(ngf x 8), LReLU	4x4x(ngf x 8)	1
4x4 ConvT(ngf x 4), LReLU	8x8x(ngf x 4)	2
4x4 ConvT(ngf x 2), LReLU	16x16x(ngf x 2)	2
4x4 ConvT(ngf x 1), LReLU	32x32x(ngf x 1)	2
4x4 ConvT(3), Tanh	64x64x3	2
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{z}$	1x1x128	-
8x8 ConvT(ngf x 8), LReLU	8x8x(ngf x 8)	1
4x4 ConvT(ngf x 4), LReLU	16x16x(ngf x 4)	2
4x4 ConvT(ngf x 2), LReLU	32x32x(ngf x 2)	2
3x3 ConvT(3), Tanh	32x32x3	1
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{z}$	1x1x128	-
4x4 ConvT(ngf x 16), LReLU	4x4x(ngf x 16)	1
4x4 ConvT(ngf x 8), LReLU	8x8x(ngf x 8)	2
4x4 ConvT(ngf x 4), LReLU	16x16x(ngf x 4)	2
4x4 ConvT(ngf x 4), LReLU	32x32x(ngf x 4)	2
4x4 ConvT(ngf x 2), LReLU	64x64x(ngf x 2)	2
4x4 ConvT(ngf x 1), LReLU	128x128x(ngfx1)	2
4x4 ConvT(3), Tanh	256x256x3	2
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{z}$	1x1x8	-
7x7 ConvT(ngf x 8), LReLU	7x7x(ngf x 8)	1
4x4 ConvT(ngf x 4), LReLU	14x14x(ngf x 4)	2
4x4 ConvT(ngf x 2), LReLU	28x28x(ngf x 2)	2
3x3 ConvT(1), Tanh	28x28x1	1

Table 3: **Network structure of the denoising diffusion network.** a) We use the sinusoidal embedding to embed the time index as in [11, 13]. b) We use the learned fourier feature module [40] to embed the input $\boldsymbol{z}$. c) The merged time embedding and context embedding is used to produce a pair of bias and scale terms to shift and scale the embedding of input $\boldsymbol{z}$. `nz` is the input dimension, as in Table 1. `nemb` is the dimension of image embedding as in Table 4.

Layers	Out size	Note
Time Embedding		
Input: t	1	time index
Sin. emb.[a]	128	
Linear, SiLU	128	
Linear	128	
Input Embedding		
Input: $\boldsymbol{z}$	nz	
Fr. emb.[b]	2x(nz)	Fourier feature
Basic Block		
Input: $\boldsymbol{z}, \boldsymbol{z}_{\text{ctx}}, \boldsymbol{z}_t$	nzf, nemb, 128	$\boldsymbol{z}$, ctx. and t emb.
Cat, SiLU	nemb + 128	merge ctx. & t emb.
Linear, SiLU	nout	
Linear	nout	Input emb.
Scale-shift[c]	nout	scale-shift $\boldsymbol{z}$ emb. w/ merged emb.
Add $\boldsymbol{z}$	nout	skip connection from $\boldsymbol{z}$
Denoising Diffusion Network		
Input: $\boldsymbol{z}, \boldsymbol{z}_{\text{ctx}}, t$	nz, nemb, 128	Input
Embedding	2x(nz), 128	Input & t emb.
Basic Block	128	Encoding
Basic Block	256	
Basic Block	256	
Basic Block	256	Intermediate
Basic Block	256	Cat & Decoding
Basic Block	128	
Basic Block	nz	Output

Hyperparameters and training details As mentioned in the main text, for the posterior and prior DAMC samplers, we set the number of diffusion steps to 100. The number of iterations in Eq. (4) is set to $M = 6$ for the experiments. The LD runs $T = 30$ and $T = 60$ iterations for posterior and prior updates during training with a step size of $s = 0.1$. For test time sampling from $\mathcal{K}_{T,\boldsymbol{z}_i|\boldsymbol{x}_i} q_{\phi_k}(\boldsymbol{z}_i|\boldsymbol{x}_i)$, we set $T = 10$ for the additional LD. For test time prior sampling of LEBM with LD, we follow [6, 31] and set $T = 100$. To further stabilize the training procedure, we i) perform gradient clipping by setting the maximal gradient norm as 100, ii) use a separate target diffusion network which is the EMA of the current diffusion network to initialize the prior and posterior updates and iii) add noise-initialized prior samples for the prior updates. These set-ups are identical across different datasets.

The parameters of all the networks are initialized with the default pytorch methods [43]. We use the Adam optimizer [44] with $\beta_1 = 0.5$ and $\beta_2 = 0.999$ to train the generator network and the energy score network. We use the AdamW optimizer [45] with $\beta_1 = 0.5$, $\beta_2 = 0.999$ and `weight_decay=1e-4` to train the diffusion network. The initial learning rates of the generator and diffusion networks are `2e-4`, and `1e-4` for the energy score network. The learning rates are decayed with a factor of 0.99 every 1K training iterations, with a minimum learning rate of `1e-5`. We run the experiments on a A6000 GPU with the batch size of 128. For GAN inversion, we reduce the batch size to 64. Training typically converges within 200K iterations on all the datasets.

Table 4: **Network structures of the encoder networks** used for the SVHN, CelebA, CIFAR-10, CelebA-HQ and MNIST (from top to bottom) datasets. For GAN inversion, the encoder network structure is the same as in [41]. Conv(n)Norm indicates a convolutional operation with n output channels followed by the Instance Normalization [42]. We use `nif=64` and `nemb=1024` for all the datasets. LReLU indicates the Leaky-ReLU activation function. The slope in Leaky ReLU is set to be 0.2.

Layers	**Out Size**	**Stride**
Input: $\boldsymbol{x}$	32x32x3	-
3x3 Conv(nif x 1)Norm, LReLU	32x32x(nif x 1)	1
4x4 Conv(nif x 2)Norm, LReLU	16x16x(nif x 2)	2
4x4 Conv(nif x 4)Norm, LReLU	8x8x(nif x 4)	2
4x4 Conv(nif x 8)Norm, LReLU	4x4x(nif x 8)	2
4x4 Conv(nemb)Norm, LReLU	1x1x(nemb)	1
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{x}$	64x64x3	-
3x3 Conv(nif x 1)Norm, LReLU	64x64x(nif x 1)	1
4x4 Conv(nif x 2)Norm, LReLU	32x32x(nif x 2)	2
4x4 Conv(nif x 4)Norm, LReLU	16x16x(nif x 4)	2
4x4 Conv(nif x 8)Norm, LReLU	8x8x(nif x 8)	2
4x4 Conv(nif x 8)Norm, LReLU	4x4x(nif x 8)	2
4x4 Conv(nemb)Norm, LReLU	1x1x(nemb)	1
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{x}$	32x32x3	-
3x3 Conv(nif x 1)Norm, LReLU	32x32x(nif x 1)	1
4x4 Conv(nif x 2)Norm, LReLU	16x16x(nif x 2)	2
4x4 Conv(nif x 4)Norm, LReLU	8x8x(nif x 4)	2
4x4 Conv(nif x 8)Norm, LReLU	4x4x(nif x 8)	2
4x4 Conv(nemb)Norm, LReLU	1x1x(nemb)	1
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{x}$	256x256x3	-
3x3 Conv(nif x 1)Norm, LReLU	256x256x(nif x 1)	1
4x4 Conv(nif x 2)Norm, LReLU	128x128x(nif x 2)	2
4x4 Conv(nif x 4)Norm, LReLU	64x64x(nif x 4)	2
4x4 Conv(nif x 4)Norm, LReLU	32x32x(nif x 4)	2
4x4 Conv(nif x 8)Norm, LReLU	16x16x(nif x 8)	2
4x4 Conv(nif x 8)Norm, LReLU	8x8x(nif x 8)	2
4x4 Conv(nif x 8)Norm, LReLU	4x4x(nif x 8)	2
4x4 Conv(nemb)Norm, LReLU	1x1x(nemb)	1
Layers	**Out Size**	**Stride**
Input: $\boldsymbol{x}$	28x28x3	-
3x3 Conv(nif x 1)Norm, LReLU	28x28x(nif x 1)	1
4x4 Conv(nif x 2)Norm, LReLU	14x14x(nif x 2)	2
4x4 Conv(nif x 4)Norm, LReLU	7x7x(nif x 4)	2
4x4 Conv(nif x 8)Norm, LReLU	3x3x(nif x 8)	2
3x3 Conv(nemb)Norm, LReLU	1x1x(nemb)	1

D Pytorch-style Pseudocode

We provide pytorch-style pseudocode to help understand the proposed method. We denote the generator network as G, the energy score network as E and the diffusion network as Q. The first page sketches the prior and posterior sampling process. The second page outlines the learning procedure.

Listing 1: Prior and posterior LD sampling.

```
def sample_langevin_prior_z(z, netE):
    s = step_size

    for i in range(n_steps):
        en = netE(z).sum()
        z_norm = 1.0 / 2.0 * torch.sum(z**2)
        z_grad = torch.autograd.grad(en + z_norm, z)[0]
        w = torch.randn_like(z)

        # Prior LD Update
        z.data = z.data - 0.5 * (s ** 2) * z_grad + s * w

    return z.detach()

def sample_langevin_posterior_z(z, x, netG, netE):
    s = step_size
    sigma_inv = 1.0 / (2.0 * sigma ** 2)

    for i in range(n_steps):
        x_hat = netG(z)
        g_log_lkhd = sigma_inv * torch.sum((x_hat - x) ** 2)

        z_n = 1.0 / 2.0 * torch.sum(z**2)
        en = netE(z).sum()

        total_en = g_log_lkhd + en + z_n
        z_grad = torch.autograd.grad(total_en, z)[0]
        w = torch.randn_like(z)

        # Posterior LD Update
        z.data = z.data - 0.5 * (s ** 2) * z_grad + s * w
    return z.detach()
```

Listing 2: Learning LEBM with DAMC.

```
for x in dataset:
    # mask for unconditional learning of DAMC
    z_mask_prob = torch.rand((len(x),), device=x.device)
    z_mask = torch.ones(len(x), device=x.device)
    z_mask[z_mask_prob < 0.2] = 0.0
    z_mask = z_mask.unsqueeze(-1)

    # draw DAMC samples
    z0 = Q(x)
    zk_pos, zk_neg = z0.detach().clone(), z0.detach().clone()

    # prior and posterior updates
    zk_pos = sample_langevin_posterior_z(
            z=zk_pos, x=x, netG=G, netE=E)
    zk_neg = sample_langevin_prior_z(
            z=torch.cat(
            [zk_neg, torch.randn_like(zk_neg)], dim=0),
            netE=E)

    # update Q
    for __ in range(6):
        Q_optimizer.zero_grad()
        Q_loss = Q.calculate_loss(
        x=x, z=zk_pos, mask=z_mask).mean()
        Q_loss.backward()
        Q_optimizer.step()

    # update G
    G_optimizer.zero_grad()
    x_hat = G(zk_pos)
    g_loss = torch.sum((x_hat - x) ** 2, dim=[1,2,3]).mean()
    g_loss.backward()
    G_optimizer.step()

    # update E
    E_optimizer.zero_grad()
    e_pos, e_neg = E(zk_pos), E(zk_neg)
    E_loss = e_pos.mean() - e_neg.mean()
    E_loss.backward()
    E_optimizer.step()
```

E Dataset and Experiment Settings

Datasets We include the following datasets to study our method: SVHN ($32 \times 32 \times 3$), CIFAR-10 ($32 \times 32 \times 3$), CelebA ($64 \times 64 \times 3$), CeleAMask-HQ (256 x 256 x 3) and MNIST (28 x 28 x 1). Following Pang et al. [6], we use the full training set of SVHN (73,257) and CIFAR-10 (50,000), and take 40,000 samples of CelebA as the training data. We take 29,500 samples from the CelebAMask-HQ dataset as the training data, and test the model on 500 held-out samples. For anomaly detection on MNIST dataset, we follow the experimental settings in [6, 31, 46, 47] and use 80% of the in-domain data to train the model. The images are scaled to $[-1, 1]$ for training.

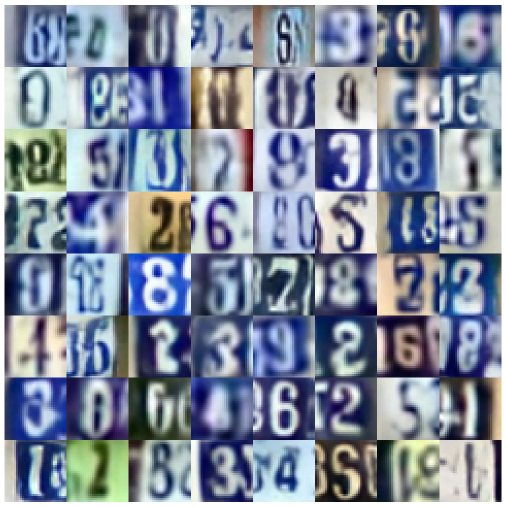
GAN inversion settings We attempt to use the DAMC sampler for GAN version on the FFHQ (256 x 256 x 3) and LSUN-Tower (256 x 256 x 3) datasets. We take 69,500 samples from the CelebAMask-HQ dataset as the training data, and use the held-out 500 samples for testing. We follow the default data splits of the LSUN dataset.

For the LEBM-based inversion method, we train a LEBM in the 14 x 512 = 7168 dim. latent space. During training, we add l_2-regularization on the energy score of LEBM to stabilize training, as suggested in [3]. For the DAMC sampler and the encoder-based inversion method [48], we generate initial posterior samples of the training data using [41] and train the DAMC sampler and the encoder-based method for 5K iterations using these samples as a warm-up step. The encoder-based method is trained by minimizing the l_2 distance between the encoder output and the target samples. After that, these two methods are trained with the default learning algorithms.

F Additional Qualitative Results

F.1 Generation

We provide additional generated samples from our models trained on SVHN (Fig. 1), CelebA (Fig. 2), CIFAR-10 (Fig. 3) and CelebA-HQ (Fig. 4).


(a) Ours-DAMC

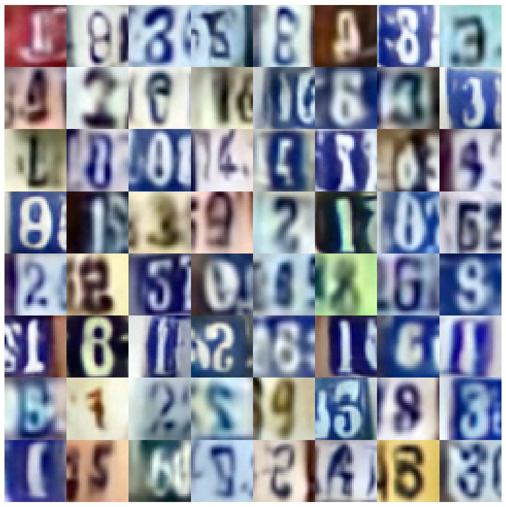
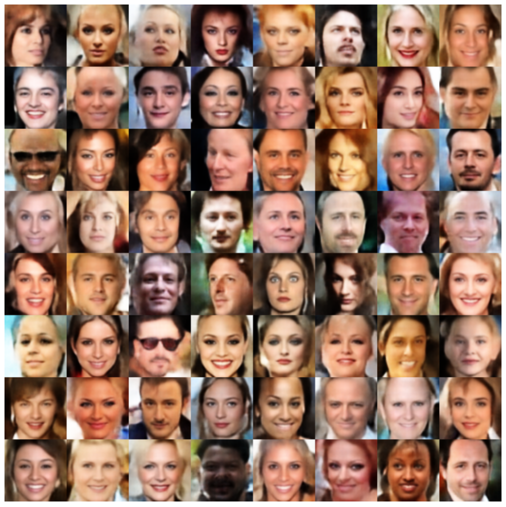

(b) Ours-LEBM

Figure 1: **Samples generated from the DAMC sampler and LEBM** trained on the SVHN dataset.


(a) Ours-DAMC

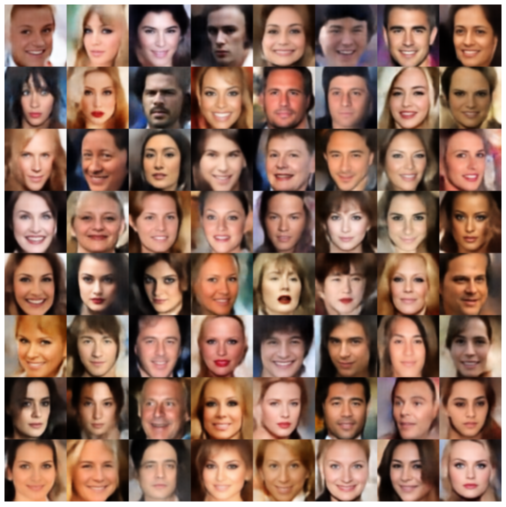
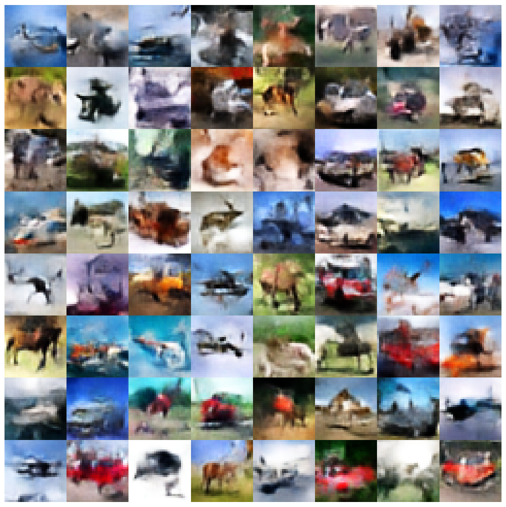

(b) Ours-LEBM

Figure 2: **Samples generated from the DAMC sampler and LEBM** trained on the CelebA dataset.

F.2 Reconstruction

We provide qualitative examples about the reconstruction results from our models trained on SVHN (Fig. 5), CelebA (Fig. 6), CIFAR-10 (Fig. 7) and CelebA-HQ (Fig. 8). Observed images are sampled from the testing set unseen during training.


(a) Ours-DAMC

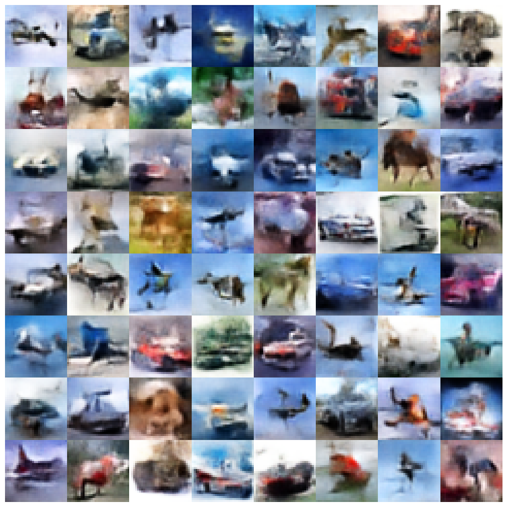
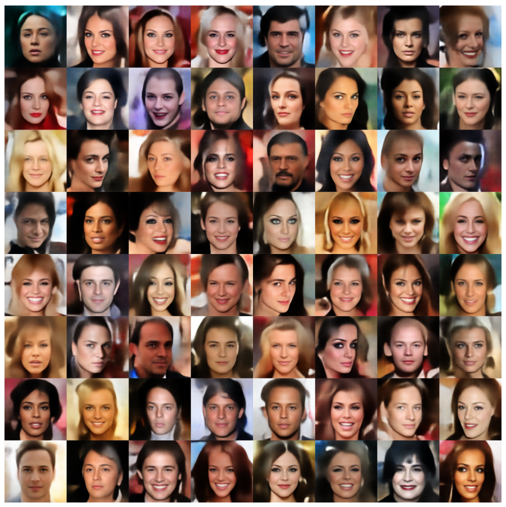

(b) Ours-LEBM

Figure 3: **Samples generated from the DAMC sampler and LEBM** trained on the CIFAR-10 dataset.


(a) Ours-DAMC

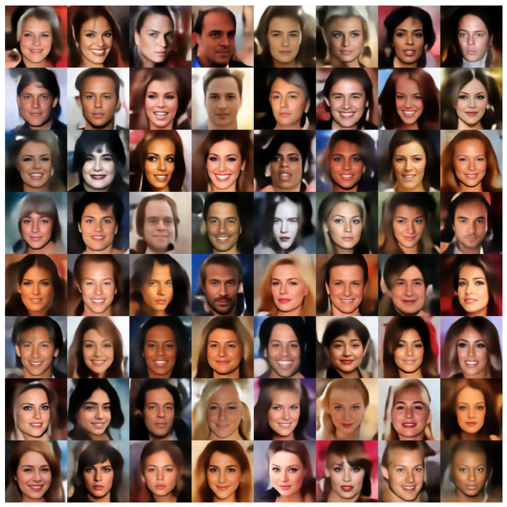
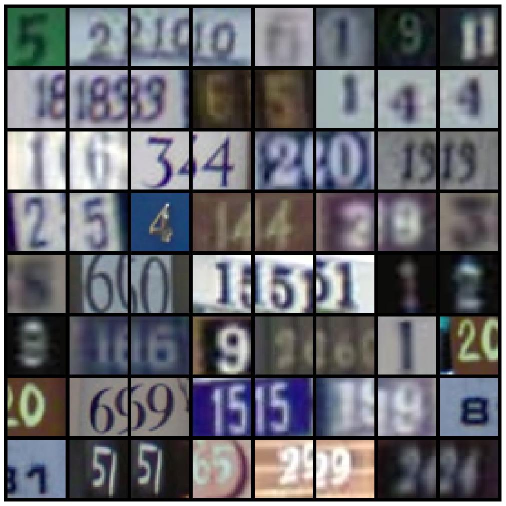

(b) Ours-LEBM

Figure 4: **Samples generated from the DAMC sampler and LEBM** trained on the CelebA-HQ dataset.

F.3 Visualization of Transitions

We provide additional visualization results of LD transitions initialized from $\mathcal{N}(0, \mathbf{I}_d)$ on SVHN (Fig. 9) and CIFAR-10 datasets (Fig. 10). For the 200-step set-up, we can see that the generation quality quickly improves by exploring the local modes with LD. For the 2500-step long-run set-up, we can see that the LD produces consistently valid results without the oversaturating issue of the long-run chain samples.


(a) Observation

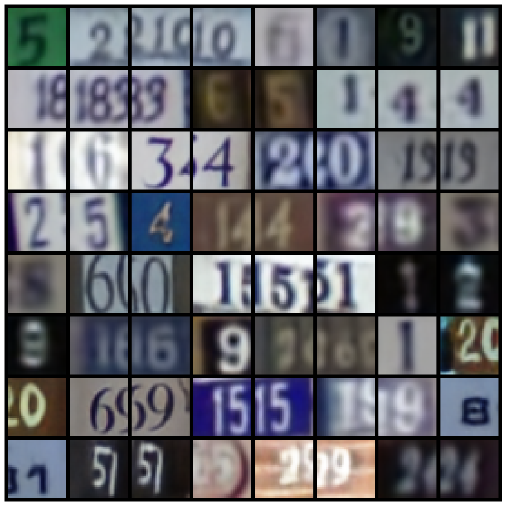
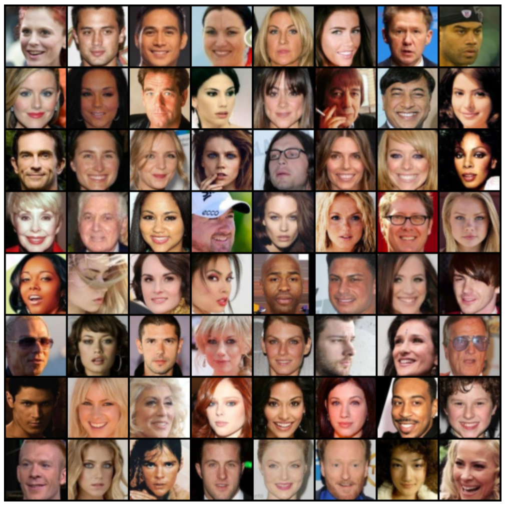

(b) Reconstruction

Figure 5: **Reconstructed samples from the posterior DAMC sampler** trained on the SVHN dataset.


(a) Observation

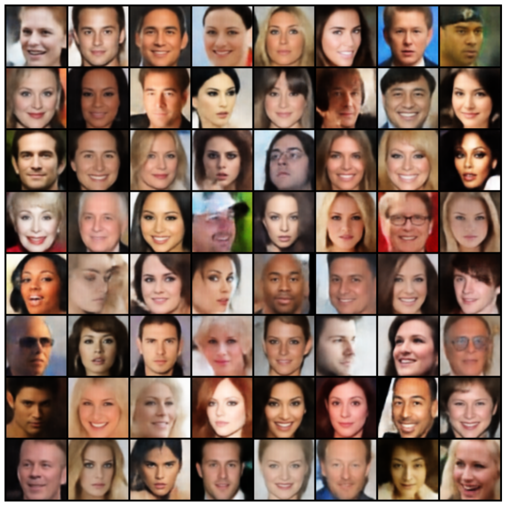
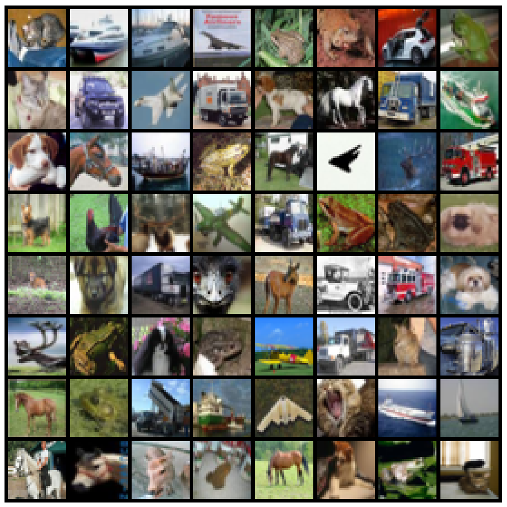

(b) Reconstruction

Figure 6: **Reconstructed samples from the posterior DAMC sampler** trained on the CelebA dataset.


(a) Observation

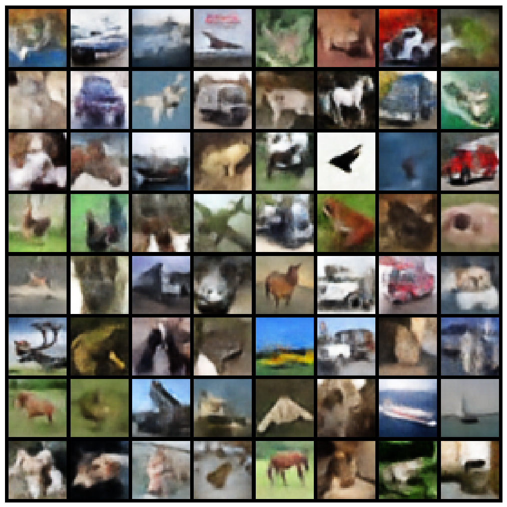
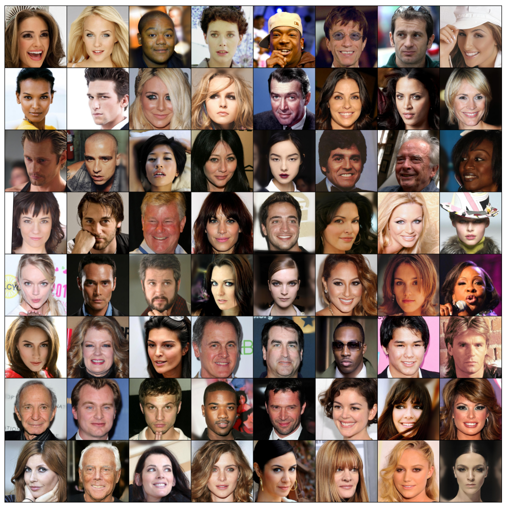

(b) Reconstruction

Figure 7: **Reconstructed samples from the posterior DAMC sampler** trained on the CIFAR-10 dataset.


(a) Observation

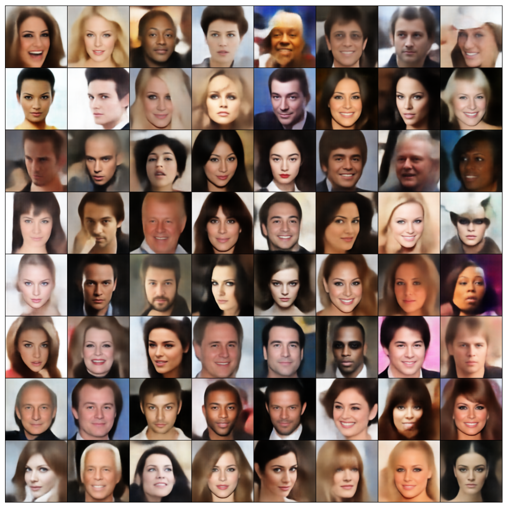
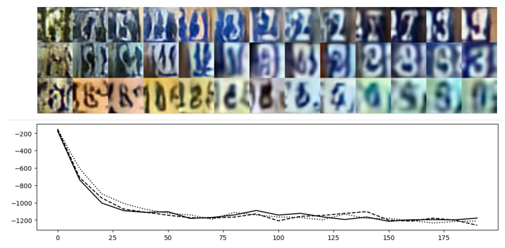

(b) Reconstruction

Figure 8: **Reconstructed samples from the posterior DAMC sampler** trained on the CelebA-HQ dataset.


(a) 200 steps

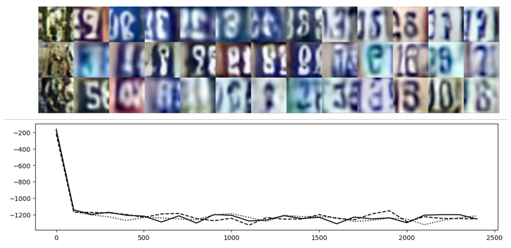
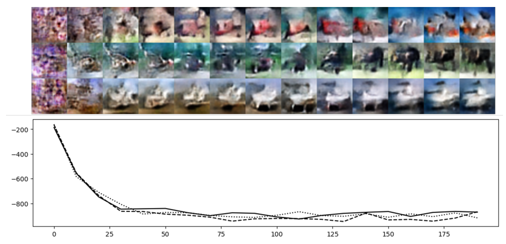
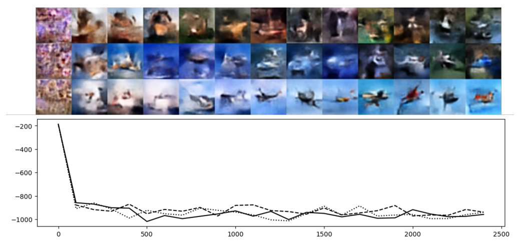

(b) 2500 steps

Figure 9: **Transition of Markov chains initialized from $\mathcal{N}(0, \mathbf{I}_d)$ towards $p_{\boldsymbol{\alpha}}(\boldsymbol{z})$ on SVHN.** We present results by running LD for 200 and 2500 steps. In each sub-figure, the top panel displays the trajectory in the data space uniformly sampled along the chain. The bottom panel shows the energy score $f_{\boldsymbol{\alpha}}(\boldsymbol{z})$ over the iterations.



(a) 200 steps



(b) 2500 steps

Figure 10: **Transition of Markov chains initialized from $\mathcal{N}(0, \mathbf{I}_d)$ towards $p_{\boldsymbol{\alpha}}(\boldsymbol{z})$ on CIFAR-10.** We present results by running LD for 200 and 2500 steps. In each sub-figure, the top panel displays the trajectory in the data space uniformly sampled along the chain. The bottom panel shows the energy score $f_{\boldsymbol{\alpha}}(\boldsymbol{z})$ over the iterations.

G Further Discussion

G.1 Limitations

We mentioned in the main text that one potential disadvantage of our method is its parameter inefficiency for introducing an extra DDPM. Although fortunately, our models are in the latent space so the network is lightweight. To be specific, on SVHN, CelebA, CIFAR-10 and CelebA-HQ datasets the number of parameters in the diffusion network is around 10% of those in the generator.

Another issue is the time efficiency. We mentioned in the main text that the time efficiency for sampling is competitive. With the batch size of 64, on these datasets the DAMC prior sampling takes $0.3s$, while 100 steps of short-run LD with LEBM takes $0.2s$. The DAMC posterior sampling takes $1.0s$, while LEBM takes $8.0s$. However, during training we need to run 30 steps of posterior LD sampling and 60 steps of prior LD sampling in each training iteration. We observe that the proposed learning method takes 15.2 minutes per training epoch, while the short-run LD-based learning method takes 14.8 minutes per epoch. These methods are slower than the VAE-base method, which takes 5.5 minutes for an training epoch. We can see that the time efficiency for training is generally bottlenecked by the LD sampling process, and could be improved in future works.

G.2 Broader Impacts

Generative models could be misused for disinformation or faking profiles. Our work focuses on the learning algorithm of energy-based prior model. Though we consider our work to be foundational and not tied to particular applications or deployments, it is possible that more powerful energy-based generative models augmented with this method may be used maliciously. Work on the reliable detection of synthetic content could be important to address such harms from generative models.

References

[1] Jianwen Xie, Yang Lu, Song-Chun Zhu, and Yingnian Wu. A theory of generative convnet. In *Proceedings of International Conference on Machine Learning (ICML)*, 2016.

[2] Erik Nijkamp, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. Learning non-convergent non-persistent short-run mcmc toward energy-based model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[3] Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models. *arXiv preprint arXiv:1903.08689*, 2019.

[4] Erik Nijkamp, Mitch Hill, Tian Han, Song-Chun Zhu, and Ying Nian Wu. On the anatomy of mcmc-based maximum likelihood learning of energy-based models. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[5] Yilun Du, Shuang Li, Joshua Tenenbaum, and Igor Mordatch. Improved contrastive divergence training of energy based models. In *Proceedings of International Conference on Machine Learning (ICML)*, 2021.

[6] Bo Pang, Tian Han, Erik Nijkamp, Song-Chun Zhu, and Ying Nian Wu. Learning latent space energy-based prior model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] Peiyu Yu, Sirui Xie, Xiaojian Ma, Yixin Zhu, Ying Nian Wu, and Song-Chun Zhu. Unsupervised foreground extraction via deep region competition. *Advances in Neural Information Processing Systems*, 2021.

[8] Peiyu Yu, Sirui Xie, Xiaojian Ma, Baoxiong Jia, Bo Pang, Ruigi Gao, Yixin Zhu, Song-Chun Zhu, and Ying Nian Wu. Latent diffusion energy-based model for interpretable text modeling. *arXiv preprint arXiv:2206.05895*, 2022.

[9] Bo Pang, Tian Han, and Ying Nian Wu. Learning latent space energy-based prior model for molecule generation. *arXiv preprint arXiv:2010.09351*, 2020.

[10] Bo Pang, Tianyang Zhao, Xu Xie, and Ying Nian Wu. Trajectory prediction with latent belief energy-based model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[11] Ruiqi Gao, Yang Song, Ben Poole, Ying Nian Wu, and Diederik P Kingma. Learning energy-based models by diffusion recovery likelihood. In *International Conference on Learning Representations (ICLR)*, 2020.

[12] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of International Conference on Machine Learning (ICML)*, 2015.

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[14] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[15] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[16] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

[17] Yingzhen Li, Richard E Turner, and Qiang Liu. Approximate inference with amortised mcmc. *arXiv preprint arXiv:1702.08343*, 2017.

[18] Jianwen Xie, Yang Lu, Ruiqi Gao, and Ying Nian Wu. Cooperative learning of energy-based model and latent variable model via mcmc teaching. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[19] Francisco Ruiz and Michalis Titsias. A contrastive divergence for combining variational inference and mcmc. In *Proceedings of International Conference on Machine Learning (ICML)*, 2019.

[20] Jianwen Xie, Zilong Zheng, Ruiqi Gao, Wenguan Wang, Song-Chun Zhu, and Ying Nian Wu. Generative voxelnet: learning energy-based models for 3d shape synthesis and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(5):2468–2484, 2020.

[21] Jianwen Xie, Zilong Zheng, and Ping Li. Learning energy-based model with variational auto-encoder as amortized sampler. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[22] Jianwen Xie, Yaxuan Zhu, Jun Li, and Ping Li. A tale of two flows: cooperative learning of langevin flow and normalizing flow toward energy-based model. *arXiv preprint arXiv:2205.06924*, 2022.

[23] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[24] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of International Conference on Machine Learning (ICML)*, 2008.

[25] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *International Conference on Learning Representations (ICLR)*, 2020.

[26] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010.

[27] Ruiqi Gao, Erik Nijkamp, Diederik P Kingma, Zhen Xu, Andrew M Dai, and Ying Nian Wu. Flow contrastive estimation of energy-based models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[28] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[29] Jyoti Aneja, Alex Schwing, Jan Kautz, and Arash Vahdat. A contrastive learning approach for training variational autoencoder priors. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[30] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in neural information processing systems*, 2020.

[31] Zhisheng Xiao and Tian Han. Adaptive multi-stage density ratio estimation for learning latent space energy-based model. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[32] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[33] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of International Conference on Machine Learning (ICML)*, 2011.

[34] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[35] JM Bernardo, MJ Bayarri, JO Berger, AP Dawid, D Heckerman, AFM Smith, M West, et al. The variational bayesian em algorithm for incomplete data: with application to scoring graphical model structures. *Bayesian statistics*, 7(453-464):210, 2003.

[36] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[39] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[40] Alexander Li and Deepak Pathak. Functional regularization for reinforcement learning via learned fourier features. *Advances in Neural Information Processing Systems*, 2021.

[41] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-domain gan inversion for real image editing. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2020.

[42] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[44] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[45] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[46] Rithesh Kumar, Sherjil Ozair, Anirudh Goyal, Aaron Courville, and Yoshua Bengio. Maximum entropy generators for energy-based models. *arXiv preprint arXiv:1901.08508*, 2019.

[47] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. *Proceedings of the 20th IEEE International Conference on Data Mining (ICDM), 2018*, 2018.

[48] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016.