On Investigating the Conservative Property of Score-Based Generative Models

Chen-Hao Chao [1] **Wei-Fang Sun** [1 2] **Bo-Wun Cheng** [1] **Chun-Yi Lee** [1]

Abstract

Existing Score-Based Models (SBMs) can be categorized into constrained SBMs (CSBMs) or unconstrained SBMs (USBMs) according to their parameterization approaches. CSBMs model probability density functions as Boltzmann distributions, and assign their predictions as the negative gradients of some scalar-valued energy functions. On the other hand, USBMs employ flexible architectures capable of directly estimating scores without the need to explicitly model energy functions. In this paper, we demonstrate that the architectural constraints of CSBMs may limit their modeling ability. In addition, we show that USBMs' inability to preserve the property of *conservativeness* may lead to degraded performance in practice. To address the above issues, we propose Quasi-Conservative Score-Based Models (QCSBMs) for keeping the advantages of both CSBMs and USBMs. Our theoretical derivations demonstrate that the training objective of QCSBMs can be efficiently integrated into the training processes by leveraging the Hutchinson's trace estimator. In addition, our experimental results on the CIFAR-10, CIFAR-100, ImageNet, and SVHN datasets validate the effectiveness of QCSBMs. Finally, we justify the advantage of QCSBMs using an example of a one-layered autoencoder.

1. Introduction

Score-Based Models (SBMs) are parameterized functions for estimating scores, which are vector fields corresponding to the gradients of log probability density functions. Based on their parameterization, SBMs can be categorized into constrained or unconstrained SBMs (Salimans & Ho, 2021). Constrained SBMs (CSBMs), also known as Energy-Based Models (EBMs), model probability density functions as Boltzmann distributions, and assign their predictions as the negative gradients of some scalar-valued energy functions (Salimans & Ho, 2021). CSBMs are able to ensure the *conservativeness* of their output vector fields. This property is essential in guaranteeing that each updates in the sampling process are determined based on the probability ratio between two consecutive sampling steps (Salimans & Ho, 2021). This, in turn, is necessary to ensure that the sample distribution converges to the true data distribution. Such a concept has been explored by the researchers of (Chen et al., 2014; Alain & Bengio, 2014; Nguyen et al., 2017; Salimans & Ho, 2021). However, the parameterization of CSBMs requires specific designs, limiting the choices of model architectures for SBMs. For example, the authors of (Vincent, 2011; Kamyshanska & Memisevic, 2013; Saremi, 2019) proposed to construct an SBM as a neural network with symmetric weights in its linear layer, which hinders its ability to be extended to more sophisticated architectures. On the other hand, the authors of (Salimans & Ho, 2021; Saremi et al., 2018; Song et al., 2019) divided an SBM into two halves: the first half explicitly parameterizes the negative energy function, while the second half is generated by automatic differentiation tools (Martens et al., 2012) to output the estimated scores. Nevertheless, these methods require that the output of the first half can only be a scalar, and the second half has to be generated using automatic differentiation tools.

In contrast, unconstrained SBMs (USBMs) employ flexible architectures capable of directly estimating the scores without explicitly modeling the energy functions. Due to their architectural flexibility, USBMs have been extensively utilized in contemporary machine learning tasks such as image generation (Song & Ermon, 2019; Ho et al., 2020; Song & Ermon, 2020; Song et al., 2021b; Nichol & Dhariwal, 2021). Among these works, Song et al. (2021b) proposed a unified framework based on a USBM, which achieved superior performance on several benchmarks. Their success demonstrated that architectural flexibility can be beneficial for SBMs. However, in spite of the empirical benefit of employing USBMs, recent research (Karras et al., 2022) suggests that the non-conservativeness of a USBM can cause

[1]Elsa Lab, Department of Computer Science, National Tsing Hua University, Taiwan. [2]NVIDIA AI Technology Center, NVIDIA Corporation. Correspondence to: Chun-Yi Lee <cylee@cs.nthu.edu.tw>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

detrimental effects on its sampling quality. In addition, our analyses in Section 3 indicate that USBMs' inability to ensure conservativeness may lead to degraded sampling performance.

To preserve both the conservativeness of CSBMs and the architectural flexibility of USBMs, we propose Quasi-Conservative Score-Based Models (QCSBMs). Instead of constraining the model architecture, QCSBMs resort to enhancing the conservativeness of USBMs through a regularization loss. Our theoretical derivations demonstrate that such a regularization term can be integrated into the training processes of SBMs efficiently through the Hutchinson's trace estimator (Hutchinson, 1989). Moreover, our experimental results showcase that the performance of Noise Conditional Score Network++ (NCSN++) (Song et al., 2021b) can be further improved by incorporating our regularization method on the CIFAR-10, CIFAR-100 (Krizhevsky & Hinton, 2009), ImageNet-32x32 (Van Oord et al., 2016), and SVHN (Netzer et al., 2011) datasets.

2. Background and Related Works

In this section, we walk through the background material and the related works for understanding the contents of this paper. We first introduce a number of score matching methods for training an SBM. Next, we describe the sampling algorithms for generating samples through an SBM. Lastly, we elaborate on the conservative property of SBMs, and the differences between CSBMs and USBMs.

2.1. Score Matching Methods

Score matching (Hyvärinen, 2005) describes the learning process to approximate the score function $\frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x})$ using a neural network $s(\cdot\,;\theta) : \mathbb{R}^D \to \mathbb{R}^D$, which is parameterized by θ and is trained through minimizing the Explicit Score Matching (ESM) objective expressed as follows:

$$\mathcal{L}_{\mathrm{ESM}}(\theta) = \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\left\| s(\boldsymbol{x};\theta) - \frac{\partial \log p(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|^2\right]. \quad (1)$$

Eq. (1) involves explicit evaluation of the true score function $\frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x})$, which is intractable for learning tasks without the true probability density function $p(\boldsymbol{x})$. To address this issue, an alternative method called Implicit Score Matching (ISM) (Hyvärinen, 2005), which excludes $\frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x})$ in the training objective, was introduced to train $s(\boldsymbol{x};\theta)$. ISM employs an equivalent loss $\mathcal{L}_{\mathrm{ISM}}$ expressed as follows:

$$\mathcal{L}_{\mathrm{ISM}}(\theta) = \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\left\| s(\boldsymbol{x};\theta)\right\|^2 + \mathrm{tr}\left(\frac{\partial s(\boldsymbol{x};\theta)}{\partial \boldsymbol{x}}\right)\right], \quad (2)$$

where $\frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)$ corresponds to the Jacobian matrix of $s(\boldsymbol{x};\theta)$, and $\mathrm{tr}\,(\cdot)$ denotes the trace of a matrix. Although $\mathcal{L}_{\mathrm{ISM}}$ avoids the evaluation of $\frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x})$, the explicit calculation of $\mathrm{tr}\left(\frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)\right)$ in Eq. (2) still requires D times of backpropagations (Song et al., 2019), which hinders $\mathcal{L}_{\mathrm{ISM}}$'s ability of being utilized in high-dimensional context. To alleviate it, a scalable objective, called Sliced Score Matching (SSM) (Song et al., 2019) loss, was proposed to approximate $\mathrm{tr}\left(\frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)\right)$ in $\mathcal{L}_{\mathrm{ISM}}$ with the Hutchinson's trace estimator (Hutchinson, 1989). Given a vector $\boldsymbol{v}$ drawn from a distribution $p_{\mathbf{v}}(\boldsymbol{v})$ satisfying $\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}\boldsymbol{v}^T\right] = I$, the Hutchinson trace estimator replaces the trace of a square matrix A with $\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^T A \boldsymbol{v}\right]$, which can be derived as:

$$\begin{aligned}\mathrm{tr}\,(A) &= \mathrm{tr}\,(AI) = \mathrm{tr}\left(A\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\boldsymbol{v}\boldsymbol{v}^T]\right) \\ &= \mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\mathrm{tr}\left(A\boldsymbol{v}\boldsymbol{v}^T\right)] = \mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\boldsymbol{v}^T A \boldsymbol{v}].\end{aligned} \quad (3)$$

The above derivation suggests that $\mathrm{tr}\left(\frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)\right)$ in Eq. (2) can be substituted with $\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\boldsymbol{v}^T \frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)\boldsymbol{v}]$, resulting in an equivalent objective $\mathcal{L}_{\mathrm{SSM}}$ expressed as follows:

$$\mathcal{L}_{\mathrm{SSM}}(\theta) = \mathbb{E}_{p(\boldsymbol{x})p_{\mathbf{v}}(\boldsymbol{v})}\left[\frac{1}{2}\left\| s(\boldsymbol{x};\theta)\right\|^2 + \boldsymbol{v}^T \frac{\partial s(\boldsymbol{x};\theta)}{\partial \boldsymbol{x}} \boldsymbol{v}\right]. \quad (4)$$

The vector-Jacobian product $\boldsymbol{v}^T \frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)$ can be calculated with a single backward propagation using automatic differentiation (Martens et al., 2012), and the expectation can be approximated using K independently sampled vectors $\{\boldsymbol{v}^{(i)}\}_{i=1}^K$. Therefore, the computation of $\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^T \frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)\boldsymbol{v}\right]$ in Eq. (4) can be less expensive than $\mathrm{tr}\left(\frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)\right)$ in Eq. (2) when $K \ll D$. The Denoising Score Matching (DSM) loss is another scalable objective formulated based on the Parzen density estimator (Vincent, 2011), which further prevents the computational overhead incurred by the gradient operation in $\mathcal{L}_{\mathrm{SSM}}$:

$$\mathcal{L}_{\mathrm{DSM}}(\theta) = \mathbb{E}_{p_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})}\left[\frac{1}{2}\left\| s(\tilde{\boldsymbol{x}};\theta) - \frac{\partial \log p(\tilde{\boldsymbol{x}}|\boldsymbol{x})}{\partial \tilde{\boldsymbol{x}}}\right\|^2\right], \quad (5)$$

where $p_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) \triangleq \frac{1}{(2\pi)^{D/2}\sigma^D} e^{\frac{-1}{2\sigma^2}\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|^2}$ is is an isotropic Gaussian smoothing kernel with a standard deviation σ, and $\frac{\partial}{\partial \tilde{\boldsymbol{x}}} \log p(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \frac{1}{\sigma^2}(\boldsymbol{x} - \tilde{\boldsymbol{x}})$. Since the computational cost of $\mathcal{L}_{\mathrm{DSM}}$ is relatively low in comparison to the other score matching losses, it has been widely adopted in contemporary modeling methods (Song & Ermon, 2019; 2020; Song et al., 2021b;a) that pursue training efficiency.

2.2. Sampling Process

Given an optimal SBM $s(\boldsymbol{x};\theta) = \frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^D$ which minimizes the score-matching objectives (i.e., Eqs. (1), (2), (4), and (5)), Langevin dynamics (Roberts & Tweedie, 1996; Roberts & Rosenthal, 1998) enables $p(\boldsymbol{x})$ to be iteratively approximated through the following equation:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \alpha s(\boldsymbol{x};\theta) + \sqrt{2\alpha}\boldsymbol{z}_t, \quad (6)$$

where α is the step size, t is the timestep, $\boldsymbol{z}_t \in \mathbb{R}^D$ is a noise vector sampled from a normal distribution $\mathcal{N}(0, I)$. Under the condition where $\alpha \to 0$ and $T \to \infty$, $\boldsymbol{x}_T$ can be generated as if it is directly sampled from $p(\boldsymbol{x})$ (Roberts & Rosenthal, 1998; Welling & Teh, 2011). Despite the theoretical guarantee of Langevin dynamics, it empirically suffers from the slow mixing issue as discussed by (Song & Ermon, 2019), which limits its ability of being utilized in practical data generation scenarios. To resolve this issue, Song et al. (2021b) proposed to extend Eq. (6) to a time-inhomogeneous variant by making the noise scale σ, the score model $s(\cdot\,;\theta)$, and step size α dependent on t. Specifically, they consider a continuous sampling process defined using a stochastic differential equation as follows:

$$d\boldsymbol{x} = [\mathbf{f}(\boldsymbol{x}, t) - g(t)^2 s(\boldsymbol{x}, t; \theta)]dt + g(t)d\bar{\boldsymbol{w}}, \quad (7)$$

where dt is an infinitesimal negative timestep, $\bar{\boldsymbol{w}}$ represents the Wiener process, $\mathbf{f}(\cdot, t)$ is the drift coefficient, and $g(t)$ is the diffusion coefficient. Contemporary score-based generation frameworks (Ho et al., 2020; Song et al., 2021b;a; Nichol & Dhariwal, 2021; Xu et al., 2022) implement such a sampling process in two different ways according to the discretization method used. One branch of them (Ho et al., 2020; Song et al., 2021b) follows the concept of Eq. (6) to discretize Eq. (7) using equal-sized steps. The other branch of them (Song et al., 2021a; Xu et al., 2022) leverages an ordinary differential equation (ODE) solver to solve the deterministic variant of Eq. (7) using adaptive step sizes.

2.3. Conservativeness and Rotation Density of a Score-Based Model

A vector field is said to be *conservative* if it can be written as the gradient of a scalar-valued function (Im et al., 2016). As proved in (Im et al., 2016), the output vector field of an SBM $s(\cdot\,;\theta)$ is said to be conservative over a smooth and simply-connected domain $\mathbb{S} \subseteq \mathbb{R}^D$ if and only if its Jacobian is symmetry for all $\boldsymbol{x} \in \mathbb{S}$, which can be equivalently expressed as the zero-rotation-density condition expressed as follows:

$$\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x}; \theta) \triangleq \frac{\partial s(\boldsymbol{x}; \theta)_i}{\partial \boldsymbol{x}_j} - \frac{\partial s(\boldsymbol{x}; \theta)_j}{\partial \boldsymbol{x}_i} = 0, \quad (8)$$

where $1 \leq i, j \leq D$, $\overline{\mathrm{ROT}_{ij}}$ is the rotation density operator (Glötzl & Richters, 2021), and $\frac{\partial}{\partial \boldsymbol{x}_j} s(\boldsymbol{x}; \theta)_i$ corresponds to the gradient of the i-th element of $s(\boldsymbol{x}; \theta)$ with respect to the j-th element of $\boldsymbol{x}$. $\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x}; \theta)$ in Eq. (8) describes the infinitesimal circulation of $s(\boldsymbol{x}; \theta)$ around $\boldsymbol{x}$.

For CSBMs, $p(\boldsymbol{x})$ is modeled as a Boltzman distribution $p(\boldsymbol{x}; \theta) = \exp(-E(\boldsymbol{x}; \theta))/Z(\theta)$, where $\exp(\cdot)$ indicates the exponential function, $E(\cdot\,;\theta) : \mathbb{R}^D \to \mathbb{R}$ represents a scalar-valued energy function, and $Z(\theta)$ refers to the partition function. Therefore, the output vector field of a CSBM can be represented as $s(\boldsymbol{x}; \theta) = \frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x}; \theta) = -\frac{\partial}{\partial \boldsymbol{x}} E(\boldsymbol{x}; \theta)$. This implies that $s(\boldsymbol{x}; \theta)$ is conservative. In other words, $s(\boldsymbol{x}; \theta)$ satisfies the zero-rotation-density condition in Eq. (8), since the mixed second derivatives of $E(\boldsymbol{x}; \theta)$ are equivalent (Alain & Bengio, 2014), which can be shown as the following:

$$\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x}; \theta) = \frac{\partial^2 E(\boldsymbol{x}; \theta)}{\partial \boldsymbol{x}_j \partial \boldsymbol{x}_i} - \frac{\partial^2 E(\boldsymbol{x}; \theta)}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_j} = 0. \quad (9)$$

Unlike CSBMs, USBMs aim to directly parameterize the true score function $\frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x})$ using a vector-valued function $s(\cdot\,;\theta) : \mathbb{R}^D \to \mathbb{R}^D$. The conservativeness of USBMs is not guaranteed, as $s(\cdot\,;\theta)$ does not necessarily correspond to the gradients of a scalar-valued function. Although it is possible to ensure the conservativeness of an USBM under an ideal scenario that $s(\boldsymbol{x}; \theta)$ perfectly minimizes the score matching target, a trained USBM typically contains approximation errors in practice. This suggests that USBMs are non-conservative in most cases, and do not satisfy the zero-rotation-density condition.

3. Motivational Examples

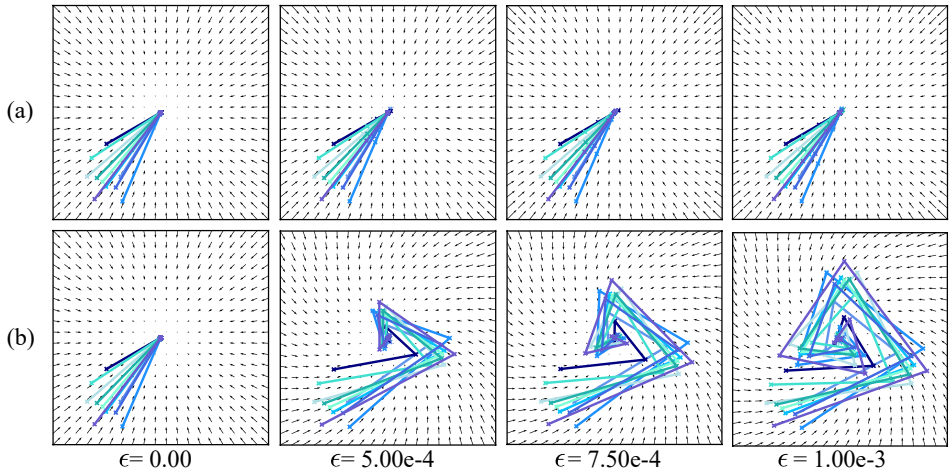
In this section, we demonstrate the importance of preserving the conservativeness as well as the architectural flexibility of SBMs. In addition, we provide the motivation behind the adoption of QCSBMs through two experiments.

3.1. The Influences of Non-Conservativeness on Sampling Process

The sampling processes described in Section 2.2 are formulated under a premise that $s(\boldsymbol{x}; \theta) = \frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x})$. In practice, however, a trained USBM contains approximation errors, which could lead to its failure in preserving the conservativeness, as discussed in Section 2.3. In this example, we inspect the impact of the non-conservativeness of USBMs on the sampling process by comparing the sampling efficiency of a USBM and a CSBM under the same approximation error ϵ, i.e., $\mathcal{L}_{\mathrm{ESM}} = \epsilon$. To quantitatively evaluate the non-conservativeness of these SBMs, we measure the magnitude of $\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x}; \theta)$ using the asymmetry metric *Asym* $\in [0, \infty)$ defined as:

$$\begin{aligned} &\mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\sum_{i,j=1}^{D}\left(\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x}; \theta)\right)^2\right] \\ &= \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\left\|J - J^T\right\|_F^2\right], \end{aligned} \quad (10)$$

where $J = \frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x})$, $\|\cdot\|_F$ is the Frobenius norm. Under a regularity condition that $p(\boldsymbol{x}) > 0, \forall \boldsymbol{x} \in \mathbb{R}^D$, *Asym* equals to 0 if and only if $s(\cdot\,;\theta)$ satisfies the zero-rotation-density condition (i.e., Proposition A.1). We also measure its normalized variant *NAsym* $\in [0, 1]$ defined as



ϵ	*Asym*	*NAsym*	Steps
0.00	0.00	0.00	1
5.00e-4	0.00	0.00	2
7.50e-4	0.00	0.00	3
1.00e-3	0.00	0.00	3
1.50e-3	0.00	0.00	4
1.75e-3	0.00	0.00	5
2.25e-3	0.00	0.00	16

ϵ	*Asym*	*NAsym*	Steps
0.00	0.00	0.00	1
5.00e-4	4.41e-7	1.77e-2	4
7.50e-4	6.60e-7	2.65e-2	5
1.00e-3	8.78e-7	3.53e-2	6
1.50e-3	1.31e-6	5.28e-2	57
1.75e-3	1.53e-6	6.15e-2	121
2.25e-3	1.96e-6	7.89e-2	238

Figure 1. The visualized examples of (a) the conservative s_C and (b) the non-conservative s_U under different choices of ϵ. The table on the right-hand side reports the results measured using the *Asym* and *NAsym* metrics as well as the number of sampling steps. For a better data visualization, the vector fields are normalized with the maximum norm of s_U and s_C in each plot.

$\mathbb{E}_{p(\boldsymbol{x})}\left[\left\|J - J^T\right\|_F^2 / (4\left\|J\right\|_F^2)\right]$ and derived in Section A.4. To evaluate the efficiency of the sampling process, we calculate the number of steps required for all sample points to move to the target during the sampling process. In this example, the USBM and the CSBM are denoted as s_{U} and s_{C}, and constructed based on Eq. (A6) presented in Appendix A.6.1.

For an illustrative purpose, we present the visualization of the sampling processes as well as the evaluation results under different choices of ϵ for two specific designs of s_{U} and s_{C} in Figs. 1 (a) and (b), respectively. As demonstrated in the visualized trajectories in Fig. 1 (b), the existence of the non-conservativeness in s_{U} incurs rotational vector fields tangent to the true score function, leading to inefficient updates during the sampling processes. In addition, the evaluation results in terms of *Asym*, *NAsym*, and the number of sampling steps further reveal that s_{U} requires more function evaluations during the sampling process than s_{C} under the same score-matching error ϵ. The above experimental evidences thus demonstrate that the non-conservativeness of a USBM may decelerate the sampling processes, which, in turn, influences the final sampling performance of it.

3.2. The Impacts of Architectural Flexibility on Modeling Ability and Sampling Performance

To ensure the conservative property of an SBM, previous works (Saremi et al., 2018; Salimans & Ho, 2021) proposed to construct the architecture such that its output vector field can be described as the gradients of a scalar-valued function. This design, however, potentially limits the modeling ability of an SBM. In this experiment, we empirically examine the influence of architectural flexibility on both the training and sampling processes. For a fair evaluation, a USBM s_{U} and a CSBM s_{C} are implemented as neural networks consisting of the same number of parameters. Following the approach described in (Salimans & Ho, 2021), these two models are represented as follows:

$$
\begin{aligned}
s_{\mathrm{U}}(\boldsymbol{x}, t; \theta_{\mathrm{U}}) &= \frac{1}{\sigma_t}(\boldsymbol{x} - f(\boldsymbol{x}, t; \theta_{\mathrm{U}})), \\
s_{\mathrm{C}}(\boldsymbol{x}, t; \theta_{\mathrm{C}}) &= -\frac{1}{2\sigma_t}\frac{\partial \left\|\boldsymbol{x} - f(\boldsymbol{x}, t; \theta_{\mathrm{C}})\right\|^2}{\partial \boldsymbol{x}},
\end{aligned}
\tag{11}
$$

where $f : \mathbb{R}^D \to \mathbb{R}^D$ is a neural network, and θ_{U} and θ_{C} are the parameters. The former is a USBM similar to that used in (Song & Ermon, 2019), while the latter corresponds to its conservative variant explored by Salimans & Ho (2021). We then compare the conservativeness, the modeling ability, and the sampling performance of both s_{U} and s_{C}, which are trained independently on three two-dimensional datasets. The conservativeness is measured using *Asym* and *NAsym*. The modeling ability of an SBM is evaluated based on its score-matching and likelihood-matching abilities, which are quantified using the score-matching error ($\mathcal{L}_{\mathrm{ESM}}$) and the negative log likelihood (NLL) metric, respectively. The sampling performance is evaluated using the Precision and Recall metrics (Kynkäänniemi et al., 2019), which measures the distances between the true samples and the generated samples based on k-nearest neighbor algorithm.

Table 1 reports the results of the above setting. The columns 'Score Error' and 'NLL' in Table 1 demonstrate that the USBMs consistently deliver better modeling performance in comparison to the CSBMs, suggesting that their architectural flexibility is indeed beneficial to the training process. On the other hand, due to the potential impact of their non-conservativeness, USBMs are unable to consistently achieve

Table 1. The evaluation results of CSBMs, USBMs, and QCSBMs in terms of their means and confidence intervals of three independent runs on the '8-Gaussian,' 'Spirals,' and 'Checkerboard' datasets, which are detailed in Appendix A.6.2. The arrow symbols ↑ / ↓ indicate that higher / lower values correspond to better performance, respectively.

Dataset	Model	*Asym* (↓)	*NAsym* (↓)	Score Error (↓)	NLL (↓)	Precision (↑)	Recall (↑)
8-Gaussian	CSBM	**0.00±0.00 e-3**	**0.00±0.00 e-3**	4.34±0.06 e-1	5.24±0.03 e+0	0.9107±0.0151	0.9057±0.0094
	USBM	1.06±0.16 e-2	1.42±0.43 e-3	**4.14±0.13 e-1**	5.02±0.03 e+0	0.9453±0.0061	0.8969±0.0191
	QCSBM	9.49±3.07 e-3	1.38±0.41 e-3	4.20±0.07 e-1	**5.01±0.09 e+0**	**0.9558±0.0022**	**0.9116±0.0154**
Spirals	CSBM	**0.00±0.00 e-1**	**0.00±0.00 e-2**	1.59±0.05 e+0	5.61±0.17 e+0	0.6221±0.0300	0.7725±0.0624
	USBM	7.35±0.41 e-1	8.19±1.82 e-2	**1.50±0.18 e+0**	5.11±0.11 e+0	0.5911±0.0573	0.8230±0.0119
	QCSBM	3.18±0.16 e-1	5.73±1.64 e-2	1.56±0.08 e+0	**5.04±0.04 e+0**	**0.6489±0.0167**	**0.8244±0.0204**
Checkerboard	CSBM	**0.00±0.00 e-2**	**0.00±0.00 e-2**	7.65±0.36 e-1	5.19±0.08 e+0	0.8990±0.0072	0.9217±0.0309
	USBM	1.05±0.18 e-1	2.51±0.43 e-2	**6.79±0.27 e-1**	5.09±0.02 e+0	0.9209±0.0089	0.9409±0.0302
	QCSBM	7.06±0.43 e-2	1.70±0.18 e-2	**6.79±0.26 e-1**	**5.08±0.02 e+0**	**0.9216±0.0027**	**0.9496±0.0156**

superior results on the precision and recall metrics, as shown in the last two columns of Table 1. The above observations thus indicate that the architectural flexibility of a USBM is crucial to its score-matching and likelihood-matching abilities. Nevertheless, its non-conservativeness may cause negative impacts on its sampling performance.

The experimental clues in Sections 3.1 and 3.2 shed light on two essential issues to be further explored and addressed. First, although USBMs benefit from their architectural flexibility, their non-conservativeness may lead to degraded sampling performance. Second, despite that CSBMs are conservative, their architectural requirement may limit their modeling abilities in practice. Based on the above observations, this paper intends to investigate a new type of SBMs, called Quasi-Conservative Score-based Models (QCSBMs), which are developed to maintain both the conservativeness as well as the architectural flexibility. As revealed in Table 1, QCSBMs are able to achieve improved results in terms of their conservativeness without sacrificing its modeling ability. In the next section, we elaborate on the formulation and implementation of QCSBMs.

4. Methodology

In this section, we introduce QCSBMs and present an efficient implementation of them. In Section 4.1, we describe the learning objective of QCSBMs, and derive its scalable variant. In Section 4.2, we detail the training procedure for QCSBMs, and discuss our implementation of its forward and backward propagation processes.

4.1. Quasi-Conservative Score-Based Models

Instead of following the concept of CSBMs to ensure the conservativeness through architectural constraints, QCSBMs resort to penalizing the non-conservativeness through a regularization loss. The training objective for QCSBMs is defined as $\mathcal{L}_{\text{Total}}$, which is expressed as follows:

$$\mathcal{L}_{\text{Total}}(\theta) = \mathcal{L}_{\text{SM}}(\theta) + \lambda \mathcal{L}_{\text{QC}}(\theta), \tag{12}$$

where $\mathcal{L}_{\text{SM}}$ can be any one of the score-matching objectives (i.e., Eqs. (1), (2), (4), or (5)), $\mathcal{L}_{\text{QC}}$ represents the regularization term reflecting the non-conservativeness, and λ is a balancing factor. As discussed in Section 3.1, the non-conservativeness of a USBM can be measured using the magnitude of its rotation densities in the Frobenius norm (i.e., Eq. (10)), suggesting a formulation of $\mathcal{L}_{\text{QC}}$ as:

$$\mathcal{L}_{\text{QC}}(\theta) = \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\left\|J - J^T\right\|_F^2\right], \tag{13}$$

where $J = \frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)$. This objective function, however, requires D times of backpropagations to explicitly calculate the Jacobian matrix of $s(\boldsymbol{x};\theta)$. In order to reduce the computational cost, we first derive an equivalent objective $\mathcal{L}_{\text{QC}}^{\text{tr}}$, and then utilize the Hutchinson's trace estimator to approximate it. The loss $\mathcal{L}_{\text{QC}}^{\text{tr}}$ is derived in Appendix A.2.2, and formulated as follows:

$$\mathcal{L}_{\text{QC}}^{\text{tr}}(\theta) = \mathbb{E}_{p(\boldsymbol{x})}\left[\text{tr}\left(JJ^T\right) - \text{tr}\left(JJ\right)\right]. \tag{14}$$

By applying the Hutchinson's trace estimator to both $\text{tr}\left(JJ^T\right)$ and $\text{tr}\left(JJ\right)$ according to Eq. (3), $\mathcal{L}_{\text{QC}}^{\text{tr}}$ can be re-expressed using an unbiased objective $\mathcal{L}_{\text{QC}}^{\text{est}}$, which is defined as the following:

$$\mathcal{L}_{\text{QC}}^{\text{est}}(\theta) = \mathbb{E}_{p(\boldsymbol{x})}\left[\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^T JJ^T \boldsymbol{v}\right] - \mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^T JJ \boldsymbol{v}\right]\right]. \tag{15}$$

Since $\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^T JJ^T \boldsymbol{v}\right]$ and $\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^T JJ \boldsymbol{v}\right]$ can be simultaneously approximated, $\mathcal{L}_{\text{QC}}^{\text{est}}$ in Eq. (15) can be rewritten as a variant $\tilde{\mathcal{L}}_{\text{QC}}^{\text{est}}$ defined as follows:

$$\tilde{\mathcal{L}}_{\text{QC}}^{\text{est}}(\theta) = \mathbb{E}_{p(\boldsymbol{x})}\left[\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^T JJ^T \boldsymbol{v} - \boldsymbol{v}^T JJ \boldsymbol{v}\right]\right]. \tag{16}$$

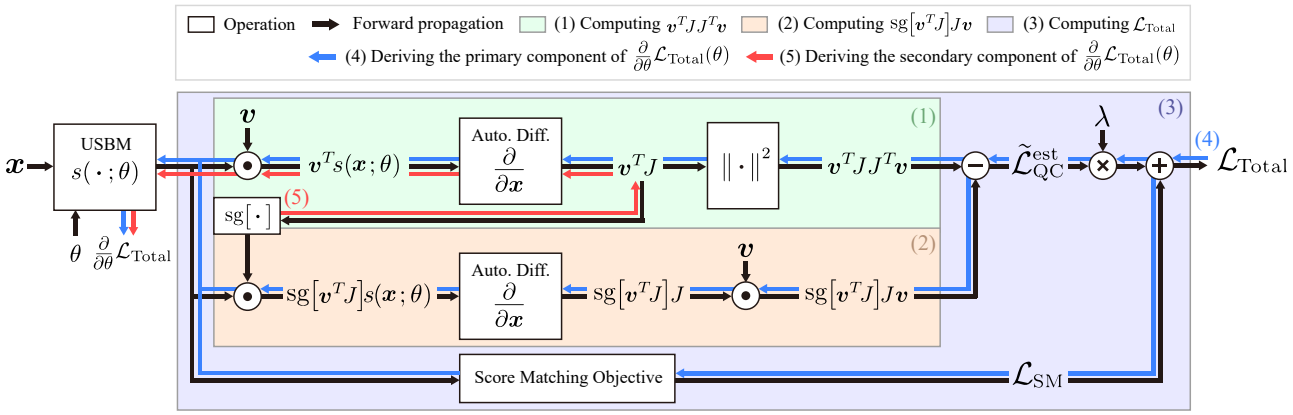


Figure 2. The computational graph of $\mathcal{L}_{\mathrm{Total}}$ in QCSBMs. The 'Auto. Diff.' blocks represent the operation of differentiating $u^T s(x;\theta)$, where u is a constant vector with respect to x.

Algorithm 1 Training Procedure of QCSBM

Input: x, v, $s(\cdot\,;\theta)$, λ

(1) Computing $v^T J J^T v$.

$v^T J \leftarrow \frac{\partial}{\partial x}[v^T s(x;\theta)]$

$v^T J J^T v \leftarrow \left\| v^T J \right\|^2$

(2) Computing $v^T J J v$.

$\mathrm{sg}\left[v^T J\right] J \leftarrow \frac{\partial}{\partial x}[\mathrm{sg}\left[v^T J\right] s(x;\theta)]$

$\mathrm{sg}\left[v^T J\right] J v \leftarrow \mathrm{sg}\left[v^T J\right] J \cdot v$

(3) Computing $\mathcal{L}_{\mathrm{Total}}(\theta)$.

$\tilde{\mathcal{L}}^{\mathrm{est}}_{\mathrm{QC}}(\theta) \leftarrow v^T J J^T v - \mathrm{sg}\left[v^T J\right] J v$

$\mathcal{L}_{\mathrm{SM}}(\theta) \leftarrow$ Eq. (1), (2), (4), or (5)

$\mathcal{L}_{\mathrm{Total}}(\theta) \leftarrow \mathcal{L}_{\mathrm{SM}}(\theta) + \lambda \tilde{\mathcal{L}}^{\mathrm{est}}_{\mathrm{QC}}(\theta)$

(4) Deriving the primary component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$.

Perform backpropagation through the blue arrows.

(5) Deriving the secondary component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$.

Perform backpropagation through the red arrows.

Update θ with $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$.

The implementation of Eq. (16) can be more efficient than that of Eq. (15) as the vector-Jacobian product $v^T J$ in Eq. (16) can be calculated once and reused in the computation of both $v^T J J^T v$ and $v^T J J v$. We provide a detailed description of such an approach in the following section.

4.2. The Training Procedure of QCSBMs

In this subsection, we walk through the proposed training procedure of QCSBMs. The training procedure is detailed in Algorithm 1, and the corresponding computational graph is illustrated in Fig. 2. For the sake of notational simplicity, we assume that both the batch size and the number of random vectors K are 1 in Algorithm 1 and Fig. 2. The entire training procedure is divided into five steps, denoted as Steps (1)∼(5), respectively. Steps (1)∼(3) describe the forward propagation process of $\mathcal{L}_{\mathrm{Total}}(\theta)$, which is depicted by the black arrows in Fig. 2. Steps (4) and (5) correspond to the backpropagation processes of the two gradient components comprising $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$, which are named the primary and secondary components, and are depicted as the blue and red arrows in Fig. 2, respectively. The detailed formulations for these two components and the rationale behind such a two-step backpropagation process are further elaborated in Appendix A.3. Please note that the symbol $\mathrm{sg}[\cdot]$ used in Algorithm 1 represents the 'stop gradient' operation, which is adopted to disconnect the computational graph.

Table 2. The NLL, *Asym*, and *NAsym* of C-NCSN++, U-NCSN++, and QC-NCSN++ evaluated on the CIFAR-10, CIFAR-100, ImageNet-32x32, and SVHN datasets.

	CIFAR-10			ImageNet-32x32		
Method	NLL	*Asym*	*NAsym*	NLL	*Asym*	*NAsym*
C-NCSN++	5.91	**0.00**	**0.00**	5.10	**0.00**	**0.00**
U-NCSN++	3.46	1.88 e8	1.90 e-3	3.96	2.05 e7	7.17 e-4
QC-NCSN++	**3.38**	3.49 e7	8.41 e-4	**3.83**	1.13 e7	5.47 e-4
	CIFAR-100			SVHN		
Method	NLL	*Asym*	*NAsym*	NLL	*Asym*	*NAsym*
C-NCSN++	5.34	**0.00**	**0.00**	5.00	**0.00**	**0.00**
U-NCSN++	3.50	2.98 e8	2.25 e-3	2.15	3.06 e7	6.54 e-4
QC-NCSN++	**3.44**	9.31 e7	1.44 e-3	**2.01**	1.69 e7	4.80 e-4

Based on the implementation described in Algorithm 1, the computation of $\tilde{\mathcal{L}}^{\mathrm{est}}_{\mathrm{QC}}$ can be more efficient than $\mathcal{L}^{\mathrm{est}}_{\mathrm{QC}}$ since it does not require repeatedly calculating the vector-Jacobian product $v^T J$ during forward propagation. Moreover, unlike $\mathcal{L}_{\mathrm{QC}}$ and $\mathcal{L}^{\mathrm{tr}}_{\mathrm{QC}}$, $\tilde{\mathcal{L}}^{\mathrm{est}}_{\mathrm{QC}}$ does not require D times of backward propagation, which justifies its computational efficiency.

5. Experiments on Real-World Datasets

In this section, we examine the effectiveness of the proposed QCSBMs on four real-world datasets: CIFAR-

Table 3. The sampling performance and NFE of C-NCSN++, U-NCSN++, and QC-NCSN++ with an ODE sampler. The arrow symbols ↑ / ↓ indicate that higher / lower values correspond to better performance, respectively.

Method	NFE (↓)	FID (↓)	IS (↑)	Prec. (↑)	Rec. (↑)
CIFAR-10					
C-NCSN++	343	16.66	7.96	0.57	0.60
U-NCSN++	170	7.48	9.24	**0.61**	**0.62**
QC-NCSN++	**124**	**7.21**	**9.25**	**0.61**	**0.62**
ImageNet-32x32					
C-NCSN++	313	24.61	8.56	0.55	0.49
U-NCSN++	148	17.09	9.80	0.55	**0.55**
QC-NCSN++	**115**	**16.62**	**9.85**	**0.56**	**0.55**
CIFAR-100					
C-NCSN++	297	21.89	7.84	0.55	0.56
U-NCSN++	168	8.95	10.09	**0.59**	0.63
QC-NCSN++	**131**	**8.90**	**10.12**	**0.59**	**0.64**
SVHN					
C-NCSN++	498	24.78	2.76	0.55	0.48
U-NCSN++	209	16.08	3.17	0.56	0.63
QC-NCSN++	**126**	**15.15**	**3.24**	**0.59**	**0.65**

Table 4. The sampling performance and NFE of C-NCSN++, U-NCSN++, and QC-NCSN++ with the PC sampler. The arrow symbols ↑ / ↓ indicate that higher / lower values correspond to better performance, respectively.

Method	NFE	FID (↓)	IS (↑)	Prec. (↑)	Rec. (↑)
CIFAR-10					
C-NCSN++	1,000	10.97	8.58	0.61	0.58
U-NCSN++		2.50	9.58	**0.67**	**0.60**
QC-NCSN++		**2.48**	**9.70**	**0.67**	**0.60**
ImageNet-32x32					
C-NCSN++	1,000	28.97	8.58	**0.61**	0.45
U-NCSN++		19.82	9.89	0.60	**0.52**
QC-NCSN++		**19.62**	**9.94**	**0.61**	**0.52**
CIFAR-100					
C-NCSN++	1,000	17.59	8.38	0.60	0.54
U-NCSN++		2.54	9.63	0.60	0.66
QC-NCSN++		**2.45**	**9.75**	**0.61**	**0.67**
SVHN					
C-NCSN++	1,000	24.71	2.66	**0.61**	0.46
U-NCSN++		14.34	3.10	0.60	**0.67**
QC-NCSN++		**13.88**	**3.12**	**0.61**	**0.67**

10, CIFAR-100 (Krizhevsky & Hinton, 2009), ImageNet-32x32 (Van Oord et al., 2016), and SVHN (Netzer et al., 2011) datasets. We employ the unconstrained architecture as well as the training procedure adopted by NCSN++ (VE) (Song et al., 2021b) as our baseline, and denote this method as 'U-NCSN++' in our experiments. On the other hand, C-NCSN++ and QC-NCSN++, which are variants of U-NCSN++ constructed based on Eq. (11) and regularized by $\tilde{\mathcal{L}}_{\mathrm{QC}}^{\mathrm{est}}$, are compared against U-NCSN++ using the NLL, *Asym*, *NAsym*, Fréchet Inception Distance (FID) (Heusel et al., 2017), and Inception Score (IS) (Barratt & Sharma, 2018), Precision, and Recall metrics. The details of the experimental setups are provided in Appendix A.6.3.

Likelihood and Conservativeness Evaluation. Table 2 reports the evaluation results of U-NCSN++, C-NCSN++, and QC-NCSN++ in terms of NLL, *Asym*, and *NAsym* on the four real-world datasets. The evaluation results of C-NCSN++ is inferior to those of U-NCSN++ and QC-NCSN++ on the NLL metric, which aligns with our observation in Section 3, suggesting that the modeling flexibility is influential to the final performance on the NLL metric. In addition, we observe that the evaluation results on the NLL metric can be further improved when $\tilde{\mathcal{L}}_{\mathrm{QC}}^{\mathrm{est}}$ is incorporated into the training process. As demonstrated in the table, QC-NCSN++, which achieves superior performance in terms of *Asym* and *NAsym* metrics, also has a noticeable improvement on the NLL metric.

Sampling with an ODE Solver. In this experiment, we examine the sampling performance of U-NCSN++, C-NCSN++, and QC-NCSN++ based on the number of function evaluations (NFE) and the FID/IS/Precision/Recall metrics. The sampler is implemented using the RK45 (Dormand & Prince, 1980) ODE solver. Table 3 presents the evaluation results of the above setting. It is observed that C-NCSN++ is inferior to U-NCSN++ and QC-NCSN++, suggesting that modeling errors can be influential to the sampling performance. On the other hand, QC-NCSN++ performs comparably to U-NCSN++ in terms of the sampling performance metrics with fewer function evaluations, indicating that QC-NCSN++ is able to deliver a better sampling efficiency.

Sampling under a Fixed NFE. In this experiment, we compare the sampling performance of C-NCSN++, U-NCSN++, and QC-NCSN++ under a fixed NFE using the Predictor-Corrector (PC) sampler (Song et al., 2021b). Different from the ODE sampler presented above, PC sampler discretizes the sampling process described Eq. (7) with equal-sized steps according to a predetermined value of NFE. Table 4 presents the evaluation results of C-NCSN++, U-NCSN++, and QC-NCSN++ when NFE is equal to 1,000. It is observed that QC-NCSN++ demonstrates improved performance in comparison to C-NCSN++ and U-NCSN++ in terms of the FID/IS/Precion/Recall metrics. The above experimental results validate the benefit of minimizing the non-conservativeness of a USBM.

The Effects of Non-Conservativeness during the Sampling Process. To further investigate the influence of non-conservativeness during the sampling process, we measure *Asym* and *NAsym* on the t-axis, i.e., the non-conservativeness under different timesteps. As shown in Fig. 4 (a), QC-NCSN++ delivers lower *Asym* and *NAsym*

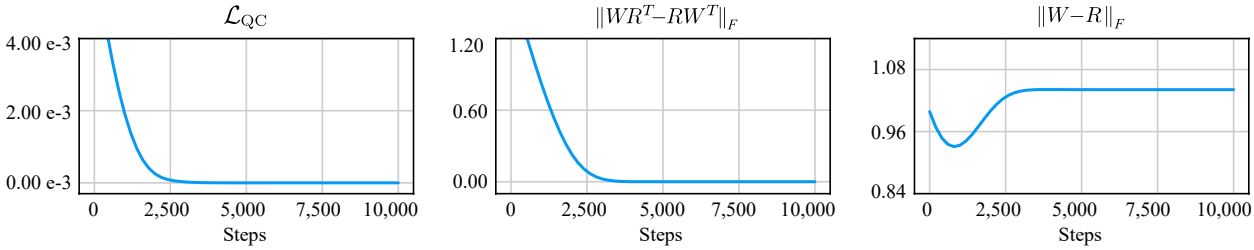


Figure 3. The trends of $\left\|WR^T - RW^T\right\|_F$ and $\|W - R\|_F$ during the minimization process of $\mathcal{L}_{\mathrm{QC}}$. The 'steps' on the x-axes refer to the training steps.

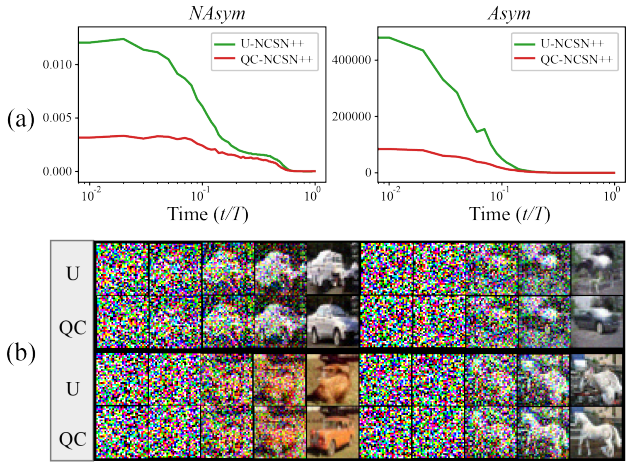


Figure 4. (a) The evaluation results of *Asym* and *NAsym* under different timestep t on the CIFAR-10 dataset. (b) Examples generated by U-NCSN++ and QC-NCSN++ with the same random seed.

under different t in comparison to U-NCSN++. This result suggests that QC-NCSN++ can be less susceptible to its non-conservativeness during the sampling process. In our qualitative comparison presented in Fig. 4 (b), we observe that some examples generated using QC-NCSN++ have noticeably improved sample quality as compared to U-NCSN++.

6. QCSBM Implemented as a One-Layered Autoencoder

A line of research (Vincent, 2011; Kamyshanska & Memisevic, 2013; Im et al., 2016; Kamyshanska & Memisevic, 2015) focuses on a type of SBM constructed as a one-layered autoencoder, in which the property of conservativeness can be systematically analyzed. Such an SBM is represented as $s(\boldsymbol{x};\theta) = Rh(W^T\boldsymbol{x} + \boldsymbol{b}) + \boldsymbol{c}$, where $h(\cdot)$ is an activation function, $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^D$, $R, W \in \mathbb{R}^{D\times H}$ are the weights of $s(\cdot\,;\theta)$ (i.e., $\theta = \{R, W, \boldsymbol{b}, \boldsymbol{c}\}$), and H is the width of the hidden-layer. As proved in (Im et al., 2016), the output vector field of $s(\cdot\,;\theta)$ is conservative if and only if $WR^T = RW^T$. To ensure the conservativeness of such an SBM, a number of works (Vincent, 2011; Kamyshanska & Memisevic, 2013; 2015) follow the concept of CSBMs and restrict the weights of $s(\cdot\,;\theta)$ to be 'tied,' i.e., $W = R$. An SBM with tied weights, however, is only a sufficient condition for its conservativeness, rather than a necessary one. This implies that there must exist some conservative $s(\cdot\,;\theta)$ that cannot be modeled using tied weights.

Instead of enforcing an SBM's weights to be tied (i.e., $W = R$), QCSBMs indirectly learn to satisfy the conservativeness condition (i.e., $WR^T = RW^T$) through minimizing $\mathcal{L}_{\mathrm{QC}}$. Fig. 3 depicts the trends of $\left\|WR^T - RW^T\right\|_F$ and $\|W - R\|_F$ during the minimization process of $\mathcal{L}_{\mathrm{QC}}$. As the training progresses, the values of $\left\|WR^T - RW^T\right\|_F$ approach zero, indicating that $s(\cdot\,;\theta)$ learns to output a conservative vector field through minimizing $\mathcal{L}_{\mathrm{QC}}$. In contrast, the values of $\|W - R\|_F$ do not decrease to zero, revealing that minimizing $\mathcal{L}_{\mathrm{QC}}$ does not necessarily lead to $W = R$. The experimental results thus suggest that QCSBMs can learn to output conservative vector fields that cannot be modeled by one-layered autoencoders with tied weights. This justifies the advantage of QCSBMs over CSBMs, as QCSBMs provide a more flexible parameterization while still maintaining their conservativeness. In Appendix A.7.1, we offer more examples to support this observation.

7. Conclusion

In this paper, we unveiled the underlying issues of CSBMs and USBMs, and highlighted the importance of preserving both of the architectural flexibility and the property of conservativeness through two motivational experiments. We proposed a new category of SBMs, named QCSBMs, in which the magnitudes of their rotation densities are minimized through a regularization loss for enhancing their property of conservativeness. We showed that such a regularization loss can be reformulated as a scalable variant based on the Hutchinson's trace estimator, and demonstrated that it can be efficiently incorporated into the training procedure of SBMs. Finally, we validated the effectiveness of QCSBMs through the experimental results on the real-world datasets, and showcased the advantage of QCSBMs over CSBMs using the example of a one-layered autoencoder.

Acknowledgements

The authors gratefully acknowledge the support from the National Science and Technology Council (NSTC) in Taiwan under grant number MOST 111-2223-E-007-004-MY3, as well as the financial support from MediaTek Inc., Taiwan. The authors would also like to express their appreciation for the donation of the GPUs from NVIDIA Corporation and NVIDIA AI Technology Center (NVAITC) used in this work. Furthermore, the authors extend their gratitude to the National Center for High-Performance Computing (NCHC) for providing the necessary computational and storage resources.

References

Alain, G. and Bengio, Y. What Regularized Auto-Encoders Learn from the Data-Generating Distribution. *The Journal of Machine Learning Research (JMLR)*, 15(1):3563–3593, 2014.

Andrilli, S. and Hecker, D. Chapter 1 - Vectors and Matrices. In *Elementary Linear Algebra (Fifth Edition)*, pp. 1–83. Academic Press, Boston, fifth edition edition, 2016.

Barratt, S. and Sharma, R. A Note on the Inception Score. In *Proc. the Theoretical Foundations and Applications of Deep Generative Models Workshop at Int. Conf. on Machine Learning (ICML)*, 2018.

Chao, C.-H., Sun, W.-F., Cheng, B.-W., Lo, Y.-C., Chang, C.-C., Liu, Y.-L., Chang, Y.-L., Chen, C.-P., and Lee, C.-Y. Denoising Likelihood Score Matching for Conditional Score-based Data Generation. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2022.

Chen, T., Fox, E., and Guestrin, C. Stochastic Gradient Hamiltonian Monte Carlo. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2014.

Dormand, J. R. and Prince, P. J. A Family of Embedded Runge-Kutta Formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

Glötzl, E. and Richters, O. Helmholtz Decomposition and Rotation Potentials in n-Dimensional Cartesian Coordinates. `arXiv:2012.13157v3 [math.PH]`, 2021.

Griewank, A. and Walther, A. Evaluating Derivatives - Principles and Techniques of Algorithmic Differentiation, Second Edition. In *Frontiers in applied mathematics*, 2000.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Ho, J., Jain, A., and Abbeel, P. Denoising Diffusion Probabilistic Models. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Hutchinson, M. F. A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Hyvärinen, A. Estimation of Non-Normalized Statistical Models by Score Matching. *Journal of Machine Learning Research (JMLR)*, 6(24):695–709, 2005.

Im, D. J., Belghazi, M. I., and Memisevic, R. Conservativeness of Untied Auto-Encoders. In *Proc. the AAAI Conf. on Artificial Intelligence (AAAI)*, 2016.

Kamyshanska, H. and Memisevic, R. On Autoencoder Scoring. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2013.

Kamyshanska, H. and Memisevic, R. The Potential Energy of an Autoencoder. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37:1261–1273, 2015.

Karras, T., Aittala, M., Aila, T., and Laine, S. Elucidating the Design Space of Diffusion-Based Generative Models. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2015.

Krizhevsky, A. and Hinton, G. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., and Aila, T. Improved Precision and Recall Metric for Assessing Generative Models. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Martens, J., Sutskever, I., and Swersky, K. Estimating the Hessian by Back-Propagating Curvature. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2012.

Naeem, M. F., Oh, S. J., Uh, Y., Choi, Y., and Yoo, J. Reliable Fidelity and Diversity Metrics for Generative Models. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2020.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading Digits in Natural Images with Unsupervised Feature Learning. 2011.

Nguyen, A. M., Clune, J., Bengio, Y., Dosovitskiy, A., and Yosinski, J. Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space. In

Proc. the Conf. on Computer Vision and Pattern Recognition (CVPR), 2017.

Nichol, A. Q. and Dhariwal, P. Improved Denoising Diffusion Probabilistic Models. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, 2019.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for Activation Functions. `arXiv:1710.05941v2 [cs.NE]`, 2017.

Roberts, G. O. and Rosenthal, J. S. Optimal Scaling of Discrete Approximations to Langevin Diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268, 1998.

Roberts, G. O. and Tweedie, R. L. Exponential Convergence of Langevin Distributions and Their Discrete Approximations. *Bernoulli*, 2(4):341–363, 1996.

Salimans, T. and Ho, J. Should EBMs Model the Energy or the Score? In *Proc. the Energy Based Models Workshop at Int. Conf. on Learning Representations (ICLR)*, 2021.

Saremi, S. On Approximating ∇f with Neural Networks. `arXiv:1910.12744v2 [stat.ML]`, 2019.

Saremi, S., Mehrjou, A., Schölkopf, B., and Hyvärinen, A. Deep Energy Estimator Networks. `arXiv:1805.08306v1 [stat.ML]`, 2018.

Song, Y. and Ermon, S. Generative Modeling by Estimating Gradients of the Data Distribution. In *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, 2019.

Song, Y. and Ermon, S. Improved Techniques for Training Score-Based Generative Models. In *Proc. Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

Song, Y., Garg, S., Shi, J., and Ermon, S. Sliced Score Matching: A Scalable Approach to Density and Score Estimation. In *Proc. the Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2019.

Song, Y., Durkan, C., Murray, I., and Ermon, S. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2021a.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-Based Generative Modeling through Stochastic Differential Equations. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2021b.

Van Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel Recurrent Neural Networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2016.

Vincent, P. A Connection between Score Matching and Denoising Autoencoders. *Neural computation*, 23(7): 1661–1674, 2011.

Welling, M. and Teh, Y. W. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *Proc. Int. Conf. on Machine Learning (ICML)*, 2011.

Xu, Y., Liu, Z., Tegmark, M., and Jaakkola, T. Poisson Flow Generative Models. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

A. Appendix

In this Appendix, we first provide the definitions for the symbols used in the main manuscript and the Appendix in Section A.1. Next, we detail the backpropagation processes described in Section 4.2, and provide the derivation for (14) in Section A.2. Then, we offer a discussion on the optimization process of QCSBMs as well as the normalized asymmetry metric in Sections A.3 and A.4, respectively. Subsequently, in Section A.5, we describe the approach to extend a QCSBM to its time-inhomogeneous variant, i.e., QC-NCSN++ described in Section 5 of the main manuscript. Finally, we provide the detailed experimental configurations in Section A.6, and a number of qualitative and quantitative experimental results in Section A.7.

A.1. List of Notations

In this section, we offer the list of notations used throughout the main manuscript and the Appendix. These notations and their descriptions are summarized in Table A1.

Symbol	Description
M	the dataset size.
D	the data dimension.
K	the number of random vectors used in the Hutchinson's trace estimator.
T	the number of discretized timesteps for the sampling algorithm.
H	the hidden dimension of the one-layered autoencoder described in Section 6.
α	the step size used in Langevin dynamics.
ϵ	the score-matching error described in Section 3.1.
σ	the standard deviation of a Gaussian distribution.
θ	the parameters of an SBM.
$\boldsymbol{x} \in \mathbb{R}^D$	a data sample.
$\tilde{\boldsymbol{x}} \in \mathbb{R}^D$	a perturbed data sample.
$\boldsymbol{z} \in \mathbb{R}^D$	a noise vector used in Langevin dynamics.
$\boldsymbol{v} \in \mathbb{R}^D$	a random vector used in the Hutchinson trace estimator.
$\boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^D$	the bias of the one-layered autoencoder described in Section 6.
$W, R \in \mathbb{R}^{D \times H}$	the weights of the one-layered autoencoder described in Section 6.
$Z(\theta)$	a partition function of a Boltzmann distribution.
$E(\cdot\,;\theta) : \mathbb{R}^D \to \mathbb{R}$	an energy model parameterized by θ.
$s(\cdot\,;\theta) : \mathbb{R}^D \to \mathbb{R}^D$	a score model parameterized by θ.
$\frac{\partial}{\partial \boldsymbol{x}} E(\boldsymbol{x};\theta)$	the gradient of $E(\boldsymbol{x};\theta)$ w.r.t. $\boldsymbol{x}$.
$\frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)$	the Jacobian matrix of $s(\boldsymbol{x};\theta)$.
J	the simplified notation for $\frac{\partial}{\partial \boldsymbol{x}} s(\boldsymbol{x};\theta)$.
$\mathcal{L}_{\mathrm{ESM}}$	Explicit Score Matching (ESM) loss defined in Eq. (1).
$\mathcal{L}_{\mathrm{ISM}}$	Implicit Score Matching loss (ISM) loss defined in Eq. (2).
$\mathcal{L}_{\mathrm{SSM}}$	Sliced Score Matching loss (SSM) loss defined in Eq. (4).
$\mathcal{L}_{\mathrm{DSM}}$	Denoising Score Matching loss (DSM) loss defined in Eq. (5).
$\mathcal{L}_{\mathrm{Total}}$	the total loss of QCSBMs defined in Eq. (12).
$\mathcal{L}_{\mathrm{QC}}$	the proposed regularization loss defined in Eq. (13).
$\mathcal{L}_{\mathrm{QC}}^{\mathrm{tr}}$	the equivalent variant of $\mathcal{L}_{\mathrm{QC}}$ defined in Eq. (14).
$\mathcal{L}_{\mathrm{QC}}^{\mathrm{est}}$	the approximated variant of $\mathcal{L}_{\mathrm{QC}}^{\mathrm{tr}}$ defined in Eq. (15).
$\tilde{\mathcal{L}}_{\mathrm{QC}}^{\mathrm{est}}$	the variant of $\mathcal{L}_{\mathrm{QC}}^{\mathrm{est}}$ defined in Eq. (16).
$\boldsymbol{u}^T\boldsymbol{v} = \boldsymbol{u} \cdot \boldsymbol{v} = \sum_i \boldsymbol{u}_i \boldsymbol{v}_i$	inner product between two vectors $\boldsymbol{u}$, $\boldsymbol{v}$.
$\mathrm{tr}\,(A) = \sum_i A_{i,i}$	trace of a matrix A.
$\|\boldsymbol{u}\| = \sqrt{\sum_i \boldsymbol{u}_i^2}$	Euclidean norm of a vector $\boldsymbol{u}$.
$\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$	Frobenius norm of a matrix A.
$\exp(\cdot)$	an exponential function.
$\mathrm{sg}\,[\cdot]$	a stop gradient operator.

Table A1. The list of symbols used in this paper.

A.2. Derivations

A.2.1. Conservativeness of a Score-based Model

Proposition A.1. *Given $p(\boldsymbol{x}) > 0, \forall \boldsymbol{x} \in \mathbb{R}^D$, Asym equals to 0 if and only if $\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x};\theta) = 0,\ \forall\, 1 \le i, j \le D$.*

Proof.

$$
\begin{aligned}
&\mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\sum_{i,j=1}^{D}\left(\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x};\theta)\right)^2\right] = 0 \\
\Leftrightarrow &\int_{\boldsymbol{x}\in\mathbb{R}^D} p(\boldsymbol{x})\frac{1}{2}\sum_{i,j=1}^{D}\left(\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x};\theta)\right)^2 d\boldsymbol{x} = 0 \\
\overset{(1)}{\Leftrightarrow} &\frac{1}{2}\sum_{i,j=1}^{D}\left(\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x};\theta)\right)^2 = 0 \\
\Leftrightarrow &\overline{\mathrm{ROT}_{ij}} s(\boldsymbol{x};\theta) = 0,\ \forall\, 1 \le i, j \le D,
\end{aligned}
$$

where (1) is due to the assumption of positiveness (i.e., $p(\boldsymbol{x}) > 0$). □

A.2.2. The Derivation of $\mathcal{L}_{\mathrm{QC}}^{\mathrm{tr}}$ in Eq. (14)

In Section 4.1, we derived the computationally efficient objective $\tilde{\mathcal{L}}_{\mathrm{QC}}^{\mathrm{est}}$ based on the equality $\mathcal{L}_{\mathrm{QC}} = \mathcal{L}_{\mathrm{QC}}^{\mathrm{tr}}$. To show that the equivalence holds, we provide a formal derivation as follows.

Proposition A.2. $\mathcal{L}_{\mathrm{QC}}(\theta) = \mathcal{L}_{\mathrm{QC}}^{\mathrm{tr}}(\theta)$.

Proof.

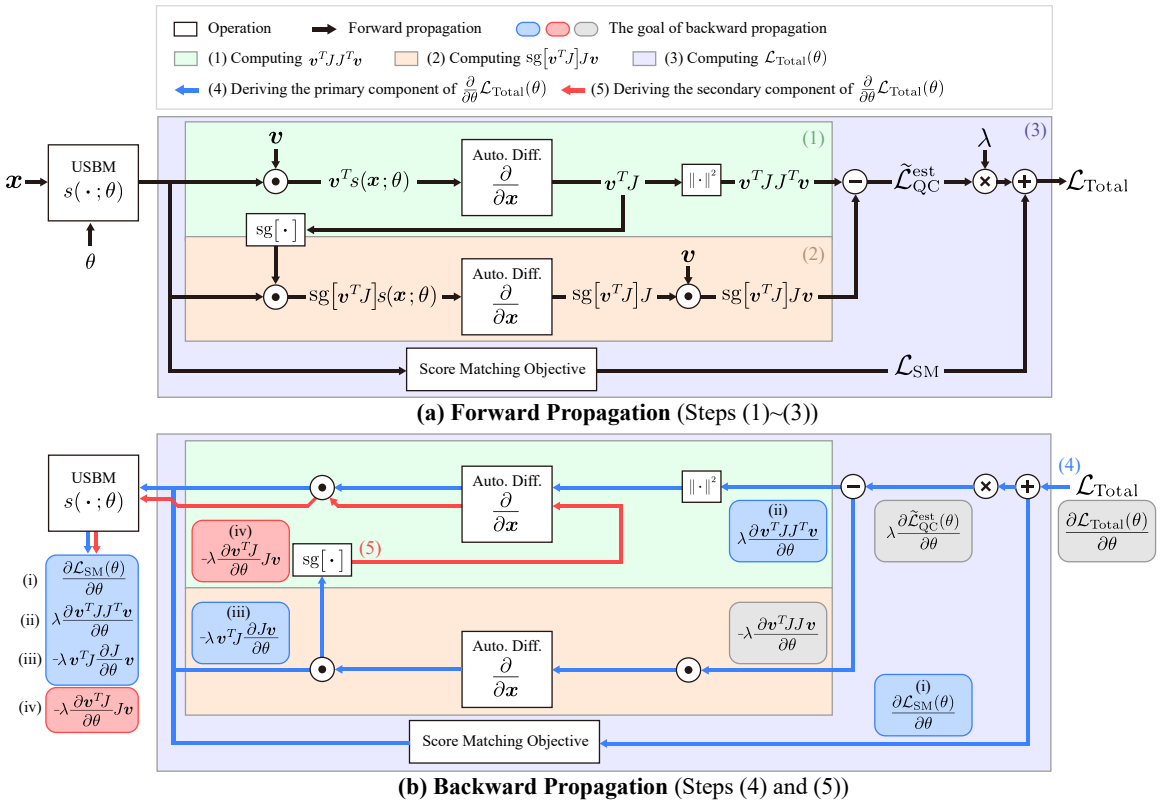
$$
\begin{aligned}
\mathcal{L}_{\mathrm{QC}}(\theta) &= \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\left\|J - J^T\right\|_F^2\right] \\
&\overset{(1)}{=} \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\mathrm{tr}\left((J - J^T)^T(J - J^T)\right)\right] \\
&= \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\mathrm{tr}\left((J^T - J)(J - J^T)\right)\right] \\
&= \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\mathrm{tr}\left(J^TJ - J^TJ^T + JJ^T - JJ\right)\right] \\
&= \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\left(\mathrm{tr}\left(J^TJ\right) - \mathrm{tr}\left(J^TJ^T\right) + \mathrm{tr}\left(JJ^T\right) - \mathrm{tr}\left(JJ\right)\right)\right] \\
&\overset{(2)}{=} \mathbb{E}_{p(\boldsymbol{x})}\left[\mathrm{tr}\left(JJ^T\right) - \mathrm{tr}\left(JJ\right)\right] \\
&= \mathcal{L}_{\mathrm{QC}}^{\mathrm{tr}}(\theta),
\end{aligned}
$$

where (1) and (2) are derived based on the properties of the trace operation, i.e., $\|A\|_F^2 = \mathrm{tr}\left(A^TA\right)$ and $\mathrm{tr}\left(AB\right) = \mathrm{tr}\left(BA\right)$, respectively. □

A.3. A Detailed Description of the Training Process of QCSBMs

The entire training procedure is divided into five steps, denoted as Steps (1)∼(5), respectively. Steps (1)∼(3) describe the forward propagation process of $\mathcal{L}_{\mathrm{Total}}(\theta)$, which is depicted by the black arrows in Fig. A1 (a). Steps (4) and (5) correspond to the backpropagation processes of the two gradient components comprising $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$, which are depicted in Fig. A1 (b). In the following paragraphs, we elaborate on the details of Steps (1)∼(5).

(1) Computing $\boldsymbol{v}^TJJ^T\boldsymbol{v}$. First, $\boldsymbol{v}^TJ$ is computed by performing backpropagation of $\boldsymbol{v}^Ts(\boldsymbol{x};\theta)$ with respect to $\boldsymbol{x}$ via automatic differentiation, which is depicted as the upper 'Auto. Diff.' block in Fig. A1 (a). Then, $\boldsymbol{v}^TJJ^T\boldsymbol{v}$ is calculated by taking the squared L2 norm on $\boldsymbol{v}^TJ$ according to the relationship: $\left\|\boldsymbol{v}^TJ\right\|^2 = \boldsymbol{v}^TJ(\boldsymbol{v}^TJ)^T = \boldsymbol{v}^TJJ^T\boldsymbol{v}$.



(a) Forward Propagation (Steps (1)~(3))

(b) Backward Propagation (Steps (4) and (5))

Figure A1. The computational graphs of $\mathcal{L}_{\mathrm{Total}}$ in QCSBMs. The upper and lower subplots depict the forward and backward propagation processes, respectively. The 'Auto. Diff.' blocks represent the operation of differentiating $\boldsymbol{u}^T s(\boldsymbol{x};\theta)$, where $\boldsymbol{u}$ is a constant vector with respect to $\boldsymbol{x}$.

(2) Computing $\boldsymbol{v}^TJJ\boldsymbol{v}$. $\mathrm{sg}\left[\boldsymbol{v}^TJ\right]s(\boldsymbol{x};\theta)$ is first calculated by taking the inner product between $\mathrm{sg}\left[\boldsymbol{v}^TJ\right]$ and $s(\boldsymbol{x};\theta)$, where the stop-gradient operator $\mathrm{sg}\left[\cdot\right]$ is applied to $\boldsymbol{v}^TJ$ to detach it from the computational graph built in Step (1). Then, $\mathrm{sg}\left[\boldsymbol{v}^TJ\right]J$ is calculated by differentiating $\mathrm{sg}\left[\boldsymbol{v}^TJ\right]s(\boldsymbol{x};\theta)$ via performing backpropagation. Stopping the gradient of $\boldsymbol{v}^TJ$ is necessary to ensure that the automatic differentiation (i.e., the lower 'Auto. Diff.' block in Fig. A1 (a)) excludes the computational graph used for differentiating $\boldsymbol{v}^TJ$, allowing $\boldsymbol{v}^TJJ$ to be correctly derived. Lastly, $\mathrm{sg}\left[\boldsymbol{v}^TJ\right]J\boldsymbol{v}$ is obtained by taking the inner product of $\mathrm{sg}\left[\boldsymbol{v}^TJ\right]J$ and $\boldsymbol{v}$.

(3) Computing $\mathcal{L}_{\mathrm{Total}}(\theta)$. Based on the results of Steps (1) and (2), $\tilde{\mathcal{L}}^{\mathrm{est}}_{\mathrm{QC}}(\theta)$ is computed by taking the expectation of $(\boldsymbol{v}^TJJ^T\boldsymbol{v} - \mathrm{sg}\left[\boldsymbol{v}^TJ\right]J\boldsymbol{v})$. Meanwhile, the score matching loss $\mathcal{L}_{\mathrm{SM}}(\theta)$ can be derived using any one of the Eqs. (1), (2), (4), and (5). Finally, $\mathcal{L}_{\mathrm{Total}}(\theta)$ is calculated by adding $\mathcal{L}_{\mathrm{SM}}(\theta)$ and $\lambda\tilde{\mathcal{L}}^{\mathrm{est}}_{\mathrm{QC}}(\theta)$, as described in Eq. (12).

(4) Deriving the primary component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$. Based on the computational graph built in Steps (1)~(3), the primary component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$ is computed by performing backward propagation through the paths in the computational graph highlighted by the blue arrows in Fig. A1 (b) using automatic differentiation. Note that these gradients are not equal to $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$ due to the adoption of the stop-gradient operator $\mathrm{sg}\left[\cdot\right]$ in Step (2). As a result, an additional secondary gradient component, which is derived in Step (5), is included to compensate it.

(5) Deriving the secondary component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$. The secondary component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$ is derived by performing backward propagation through the paths in the computational graph highlighted by the red arrows in Fig. A1 (b) using automatic differentiation. By accumulating the gradients of the primary and the secondary components, the gradients $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$ can be correctly calculated.

The Derivation of the primary and secondary components of $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$. In Steps (4) and (5), we decompose $\frac{\partial}{\partial\theta}\mathcal{L}_{\mathrm{Total}}(\theta)$ as the primary and secondary components, and separately derive them. To further elaborate on such a backward propagation process, we offer a detailed description in this subsection. For the sake of notational simplicity, we

assume that both the batch size and the number of random vectors K are 1.

According to the rule of sum and the rule of product from vector calculus, the gradient of the total loss $\frac{\partial}{\partial\theta}\mathcal{L}_{\text{Total}}(\theta)$ can be decomposed as the sum of $\frac{\partial}{\partial\theta}\mathcal{L}_{\text{SM}}(\theta)$, $\lambda\frac{\partial}{\partial\theta}\boldsymbol{v}^TJJ^T\boldsymbol{v}$, $-\lambda(\frac{\partial}{\partial\theta}\boldsymbol{v}^TJ)J\boldsymbol{v}$, and $-\lambda\boldsymbol{v}^TJ(\frac{\partial}{\partial\theta}J\boldsymbol{v})$, indexed as (i)∼(iv) respectively. The derivation is shown as the following:

$$\begin{aligned}
\frac{\partial\mathcal{L}_{\text{Total}}(\theta)}{\partial\theta} &= \frac{\partial(\mathcal{L}_{\text{SM}}(\theta)+\lambda\mathcal{L}^{\text{est}}_{\text{QC}}(\theta))}{\partial\theta} \\
&= \frac{\partial\mathcal{L}_{\text{SM}}(\theta)}{\partial\theta}+\lambda\frac{\partial\mathcal{L}^{\text{est}}_{\text{QC}}(\theta)}{\partial\theta} \\
&= \frac{\partial\mathcal{L}_{\text{SM}}(\theta)}{\partial\theta}+\lambda\frac{\partial(\boldsymbol{v}^TJJ^T\boldsymbol{v}-\boldsymbol{v}^TJJ\boldsymbol{v})}{\partial\theta} \\
&= \frac{\partial\mathcal{L}_{\text{SM}}(\theta)}{\partial\theta}+\lambda\frac{\partial\boldsymbol{v}^TJJ^T\boldsymbol{v}}{\partial\theta}-\lambda\frac{\partial\boldsymbol{v}^TJJ\boldsymbol{v}}{\partial\theta} \\
&= \underbrace{\frac{\partial\mathcal{L}_{\text{SM}}(\theta)}{\partial\theta}}_{\text{(i)}}+\underbrace{\lambda\frac{\partial\boldsymbol{v}^TJJ^T\boldsymbol{v}}{\partial\theta}}_{\text{(ii)}}+\underbrace{(-\lambda)\boldsymbol{v}^TJ\frac{\partial J\boldsymbol{v}}{\partial\theta}}_{\text{(iii)}}+\underbrace{(-\lambda)\frac{\partial\boldsymbol{v}^TJ}{\partial\theta}J\boldsymbol{v}}_{\text{(iv)}}.
\end{aligned}$$

We name the sum of (i)∼(iii) the *primary* component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\text{Total}}(\theta)$, and the term (iv) the *secondary* component of $\frac{\partial}{\partial\theta}\mathcal{L}_{\text{Total}}(\theta)$. Such a decomposition suggests that $\frac{\partial}{\partial\theta}\mathcal{L}_{\text{Total}}(\theta)$ can be separately computed based on the computational graph built in Steps (1)∼(3) as shown in the upper subplot of Fig. A1. For the primary component, the sum of (i)∼(iii) is computed by performing backward propagation through the paths in the computational graph highlighted by the blue arrows in the lower subplot of Fig. A1 using automatic differentiation. For the secondary component, the term (iv) is calculated by performing backward propagation through the red arrows in the lower subplot of Fig. A1. Through these two steps, $\frac{\partial}{\partial\theta}\mathcal{L}_{\text{Total}}(\theta)$ can be correctly derived.

A.4. Normalized Asymmetry Metric

In this section, we elaborate on the formulation of the normalized asymmetry metric *NAsym* and derive a computationally efficient implementation of it using the Hutchinson's trace estimator.

Derivation of the *NAsym* Metric. As described in (Andrilli & Hecker, 2016), any matrix A can be uniquely decomposed into a symmetric matrix A_{sym} and a skew-symmetric matrix A_{skew} as follows:

$$A = A_{\text{sym}} + A_{\text{skew}} = \frac{A+A^T}{2}+\frac{A-A^T}{2}. \tag{A1}$$

Based on Eq. (A1), the Jacobian J of a USBM $s(\cdot\,;\theta)$ can be written as the sum of a symmetric matrix $J_{\text{sym}} = (J+J^T)/2$ and a skew-symmetric matrix $J_{\text{skew}} = (J-J^T)/2$. Under such a definition, the *NAsym* metric introduced in Section 3.1 can be formulated as follows:

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\frac{\|J_{\text{skew}}\|_F^2}{\|J\|_F^2}\right] = \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{\left\|\frac{1}{2}(J-J^T)\right\|_F^2}{\|J\|_F^2}\right] = \mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{4}\frac{\left\|J-J^T\right\|_F^2}{\|J\|_F^2}\right]. \tag{A2}$$

This metric measures the ratio of the squared Frobenius norm of the skew-symmetric matrix $\|J_{\text{skew}}\|_F^2$ to the squared Frobenius norm of the Jacobian matrix $\|J\|_F^2$, and falls within the range $[0,1]$. *NAsym* $= 1$ corresponds to the condition where J_{skew} dominates J, implying that J is skew-symmetric. On the contrary, *NAsym* $= 0$ indicates that J only contains the symmetric component J_{sym}, suggesting that J is symmetric. Since the squared Frobenius norm of the skew-symmetric matrix can be written as the sum of the squared rotation densities of $s(\boldsymbol{x};\theta)$, i.e., $\|J_{\text{skew}}\|_F^2 = \left\|(J-J^T)/2\right\|_F^2 = \frac{1}{4}\sum_{i,j=1}^{D}(\frac{\partial}{\partial\boldsymbol{x}_j}s(\boldsymbol{x};\theta)_i - \frac{\partial}{\partial\boldsymbol{x}_i}s(\boldsymbol{x};\theta)_j)^2 = \frac{1}{4}\sum_{i,j=1}^{D}\left(\overline{\text{ROT}_{ij}}s(\boldsymbol{x};\theta)\right)^2$, *NAsym* can be adopted to measure the non-conservativeness of $s(\cdot\,;\theta)$, as mentioned in Section 3.1.

An Efficient Implementation of *NAsym*. Since Eq. (A2) involves the explicit calculation of the Jacobian matrix J, evaluating the *NAsym* metric for any single instance requires D times of backward propagations. This indicates that the evaluation cost could grow significantly when D becomes large. To reduce the evaluation cost, we utilize the Hutchinson's

trace estimator to approximate the *NAsym* metric based on the following derivation:

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{4}\frac{\left\|J-J^T\right\|_F^2}{\|J\|_F^2}\right]=\mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\frac{\operatorname{tr}\left(JJ^T\right)-\operatorname{tr}\left(JJ\right)}{\operatorname{tr}\left(JJ^T\right)}\right]$$
$$=\mathbb{E}_{p(\boldsymbol{x})}\left[\frac{1}{2}\frac{\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\boldsymbol{v}^TJJ^T\boldsymbol{v}]-\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\boldsymbol{v}^TJJ\boldsymbol{v}]}{\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\boldsymbol{v}^TJJ^T\boldsymbol{v}]}\right]=\mathbb{E}_{p(\boldsymbol{x})p_{\mathbf{v}}(\boldsymbol{v})}\left[\frac{1}{2}\frac{\boldsymbol{v}^TJJ^T\boldsymbol{v}-\boldsymbol{v}^TJJ\boldsymbol{v}}{\boldsymbol{v}^TJJ^T\boldsymbol{v}}\right]. \tag{A3}$$

The expectation $\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}[\cdot]$ can be approximated using K random vectors. In addition, the terms $\boldsymbol{v}^TJJ^T\boldsymbol{v}$ and $\boldsymbol{v}^TJJ\boldsymbol{v}$ in Eq. (A3) can be efficiently calculated based on Steps (1) and (2) described in Section 4.2. This suggests that the computational cost of evaluating *NAsym* can be significantly reduced when $K \ll D$.

A.5. Time-Inhomogeneous QCSBMs

In this section, we demonstrate how a QCSBM is converted to its time-inhomogeneous variant QC-NCSN++, which was described in Section 5 of the main manuscript. We first explain the modifications made in the sampling process. Then, we elaborate on the corresponding adjustments in the score-matching objective and the regularization loss.

Sampling Process. QC-NCSN++ adopts the variance exploding (VE) diffusion process identical to that employed in NCSN++ (VE) (Song et al., 2021b), which is a time-inhomogeneous sampling algorithm. In this sampling algorithm, the SBM and the step size are respectively represented as $s(\cdot\,;\theta,\sigma_t)$ and $\alpha_t=\frac{\partial}{\partial t}\sigma_t^2$, where σ_t is a time-dependent standard deviation. In C-NCSN++, U-NCSN++, and QC-NCSN++, σ_t is set to $\sigma_{\min}(\sigma_{\min}/\sigma_{\max})^{\frac{t}{T}}$ (Song et al., 2021b), where T is the total number of timesteps in the sampling process, $\sigma_{\min}$ is a constant representing a minimal noise scale, and $\sigma_{\max}$ is a constant denoting a maximal noise scale.

Training Objectives. Since the above time-inhomogeneous sampling process requires the SBM $s(\cdot\,;\theta,\sigma_t)$ to be conditioned on a time-dependent standard deviation σ_t, the training objectives of $s(\cdot\,;\theta,\sigma_t)$ have to be modified accordingly. For example, the score-matching objective $\mathcal{L}_{\mathrm{DSM}}$ used in C-NCSN++, U-NCSN++, and QC-NCSN++ is modified as follows:

$$\mathbb{E}_{\mathcal{U}(t)}\left[\lambda(t)\mathbb{E}_{p_{\sigma_t}(\tilde{\boldsymbol{x}}|\boldsymbol{x})p_0(\boldsymbol{x})}\left[\left\|s(\tilde{\boldsymbol{x}};\theta,\sigma_t)-\frac{\partial\log p_{\sigma_t}(\tilde{\boldsymbol{x}}|\boldsymbol{x})}{\partial\tilde{\boldsymbol{x}}}\right\|^2\right]\right], \tag{A4}$$

where $\mathcal{U}(t)$ is a uniform distribution defined on the interval $[0,T]$, and $\lambda(t)$ is a time-dependent coefficient for balancing the loss functions of different t. Meanwhile, the regularization term $\tilde{\mathcal{L}}_{\mathrm{QC}}^{\mathrm{est}}$ used in QC-NCSN++ is adjusted according to $\lambda(t)$, which is formulated as follows:

$$\mathbb{E}_{\mathcal{U}(t)}\left[\lambda(t)\mathbb{E}_{p_{\sigma_t}(\tilde{\boldsymbol{x}}|\boldsymbol{x})p_0(\boldsymbol{x})}\left[\mathbb{E}_{p_{\mathbf{v}}(\boldsymbol{v})}\left[\boldsymbol{v}^TJJ^T\boldsymbol{v}-\boldsymbol{v}^TJJ\boldsymbol{v}\right]\right]\right], \tag{A5}$$

where $J=\frac{\partial}{\partial\tilde{\boldsymbol{x}}}s(\tilde{\boldsymbol{x}};\theta,\sigma_t)$.

A.6. Experimental Setups

In this section, we elaborate on the experimental configurations and provide the detailed hyperparameter setups for the experiments presented in Sections 3 and 5 of the main manuscript. The code implementation for the experiments is provided in the following repository: `https://github.com/chen-hao-chao/qcsbm`.

A.6.1. Experimental Setups for Section 3.1

In Section 3.1, we compare the sampling efficiency of a USBM and a CSBM with an approximation error ϵ. These SBMs are formulated based on the following equation:

$$s(\boldsymbol{x})=\frac{\partial}{\partial\boldsymbol{x}}\log p(\boldsymbol{x})+\sqrt{\frac{2\epsilon\mu(\boldsymbol{x})}{\|\boldsymbol{x}\|^2\,p(\boldsymbol{x})}}\boldsymbol{u}(\boldsymbol{x}), \tag{A6}$$

where p is the target distribution, μ is an arbitrary distribution, and $\boldsymbol{u}(\boldsymbol{x})\in\mathbb{R}^D$ is a vector function with its norm equal to the norm of its input (i.e., $\|\boldsymbol{u}(\boldsymbol{x})\|=\|\boldsymbol{x}\|$). To show that the SBM $s(\cdot)$ in Eq. (A6) satisfies $\mathcal{L}_{\mathrm{ESM}}=\epsilon$, we provide the following proposition.

Proposition A.3. *Given $\epsilon > 0$, a target distribution p, and an arbitrary pdf μ, s defined in Eq. (A6) satisfies $\mathcal{L}_{\mathrm{ESM}} = \epsilon$.*

Proof.

$$\begin{aligned}
\mathcal{L}_{\mathrm{ESM}} &= \int_{\boldsymbol{x}} p(\boldsymbol{x}) \frac{1}{2} \left\| s(\boldsymbol{x}) - \frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x}) \right\|^2 d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} p(\boldsymbol{x}) \frac{1}{2} \left\| \frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x}) + \sqrt{\frac{2\epsilon\mu(\boldsymbol{x})}{\|\boldsymbol{x}\|^2 p(\boldsymbol{x})}} \boldsymbol{u}(\boldsymbol{x}) - \frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x}) \right\|^2 d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} p(\boldsymbol{x}) \frac{1}{2} \left\| \sqrt{\frac{2\epsilon\mu(\boldsymbol{x})}{\|\boldsymbol{x}\|^2 p(\boldsymbol{x})}} \boldsymbol{u}(\boldsymbol{x}) \right\|^2 d\boldsymbol{x} = \int_{\boldsymbol{x}} p(\boldsymbol{x}) \frac{1}{2} \frac{2\epsilon\mu(\boldsymbol{x})}{\|\boldsymbol{x}\|^2 p(\boldsymbol{x})} \|\boldsymbol{u}(\boldsymbol{x})\|^2 d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} \mu(\boldsymbol{x}) \frac{\epsilon \|\boldsymbol{u}(\boldsymbol{x})\|^2}{\|\boldsymbol{x}\|^2} d\boldsymbol{x} = \int_{\boldsymbol{x}} \mu(\boldsymbol{x}) \epsilon d\boldsymbol{x} = \epsilon
\end{aligned}$$

□

In the motivational example presented in Section 3.1, we choose $p = \mathcal{N}(0; \sigma^2 I)$, and select $\mu = \frac{1}{10}\sum_{i=1}^{10} \mathcal{N}([3\cos(\frac{2i\pi}{10}), 3\sin(\frac{2i\pi}{10})]^T; I)$. We consider $\boldsymbol{u}(\boldsymbol{x}) = [-x_2, x_1]^T$ for s_{U}, and $\boldsymbol{u}(\boldsymbol{x}) = [x_1, x_2]^T$ for s_{C}, where $\boldsymbol{x} = [x_1, x_2]^T$. In particular, $[-x_2, x_1]^T$ is a rotational vector field with each vector tangent to the true score function $\frac{\partial}{\partial \boldsymbol{x}} \log p(\boldsymbol{x}) = -1/\sigma^2 [x_1, x_2]^T$. On the other hand, $[x_1, x_2]^T$ is a vector field with each vector pointing to the opposite direction against the true score function. For an illustrative purpose, we leverage a deterministic variant of Eq. (6) (i.e., $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \alpha_t s(\boldsymbol{x}_t)$) as our sampler to generate samples based on s_{U} and s_{C}, and calculate the steps required for all samples to move to the center of p.

A.6.2. Experimental Setups for the Motivational Examples

Datasets. The motivational experiments in Section 3.2 are performed on the 8-Gaussian, Spirals, and Checkerboard datasets as shown in Fig. A2 (a). The data points of the 8-Gaussian dataset are sampled from eight separate Gaussian distributions centered at $(\cos(\frac{\pi w}{4}), \sin(\frac{\pi w}{4}))$, where $w \in \{1, ..., 8\}$. The data points of the Spirals dataset are sampled from two separate curves $(-\pi\sqrt{w}\cos(\pi\sqrt{w}), \pi\sqrt{w}\sin(\pi\sqrt{w}))$ and $(\pi\sqrt{w}\cos(\pi\sqrt{w}), -\pi\sqrt{w}\sin(\pi\sqrt{w}))$, where $w \in [0, 1]$. Lastly, the data points of the Checkerboard dataset are sampled from $(4w - 2, t - 2s + \lfloor 4w - 2 \rfloor \bmod 2)$, where $w \in [0, 1]$, $t \in [0, 1]$, $s \in \{0, 1\}$, $\lfloor \cdot \rfloor$ is a floor function, and mod is the modulo operation.

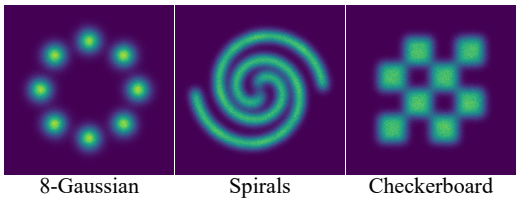


Figure A2. The visualizations of the 8-Gaussian, Spirals, and Checkerboard datasets.

Training and Implementation Details. The network architecture of f is a three-layered multilayer perceptron (MLP) with $(64, 128, 256)$ neurons and Swish (Ramachandran et al., 2017) as its activation function. This model architecture is similar to that used in the two-dimensional experiments of (Chao et al., 2022). The SBMs s_{U} and s_{C} are trained using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 7.5×10^{-4} and a batch size of $5,000$. The balancing factor λ is fixed to 0.1. The maximal and minimal noise scales $\sigma_{\max}$ and $\sigma_{\min}$ are set to 3 and 0.1, respectively.

Evaluation Method. The asymmetry (*Asym*) and normalized asymmetry (*NAsym*) metrics are evaluated on T discretized timesteps according to the following equations:

$$\frac{1}{TM} \sum_{t=1}^{T} \sum_{m=1}^{M} \mathrm{tr}\left(JJ^T\right) - \mathrm{tr}\left(JJ\right), \tag{A7}$$

$$\frac{1}{TM} \sum_{t=1}^{T} \sum_{m=1}^{M} \frac{\mathrm{tr}\left(JJ^T\right) - \mathrm{tr}\left(JJ\right)}{\mathrm{tr}\left(JJ^T\right)}, \tag{A8}$$

where $J = \frac{\partial}{\partial \tilde{\mathbf{x}}_m^{(t)}} s(\tilde{\mathbf{x}}_m^{(t)}; \theta, \sigma_t)$, $\{\tilde{\mathbf{x}}_m^{(t)}\}_{m=1}^{M}$ represents a set of testing data points, and M denotes the size of the testing set.

On the other hand, the score error is evaluated based on the following formula:

$$\frac{1}{TM}\sum_{t=1}^{T}\sum_{m=1}^{M}\left\| s(\tilde{\mathbf{x}}_m^{(t)};\theta,\sigma_t) - \frac{\partial \log p_{\sigma_t}(\tilde{\mathbf{x}}_m^{(t)})}{\partial \tilde{\mathbf{x}}_m^{(t)}} \right\|^2, \tag{A9}$$

where $p_{\sigma_t}(\tilde{\boldsymbol{x}}) = \int p_{\sigma_t}(\tilde{\boldsymbol{x}}|\boldsymbol{x})p_0(\boldsymbol{x})d\boldsymbol{x}$ and its closed form is derived according to (Chao et al., 2022). In our implementation, T and M are set as 10 and $5,000$, respectively.

A.6.3. Experimental Setups for the Evaluations on the Real-World Datasets

Datasets. The experiments presented in Section 5 are performed on the CIFAR-10, CIFAR-100 (Krizhevsky & Hinton, 2009), ImageNet-32x32 (Van Oord et al., 2016), and SVHN (Netzer et al., 2011) datasets. The training and test sets of Cifar-10 and Cifar-100 contain 50,000 and 10,000 images, respectively. The training and test sets of SVHN contain 73,257 and 26,032 images, respectively. On the other hand, the training and the test sets of ImageNet-32x32 consist of 1,281,149 and 49,999 images, respectively.

Training and Implementation Details. C-NCSN++, U-NCSN++, and QC-NCSN++ are implemented using the `Pytorch` framework. C-NCSN++, U-NCSN++, and QC-NCSN++ are trained using the Adam optimizer with a learning rate of 2×10^{-4}. The training procedure of U-NCSN++ requires 600,000 iterations for convergence for the CIFAR-10, CIFAR-100, and ImageNet-32x32 datasets, while it requires 300,000 iterations for convergence for the SVHN dataset. On the other hand, the optimization of C-NCSN++ is terminated at the early stage of the training process, since we observed that the sampling process using ODE sampler of C-NCSN++ optimized according to the aforementioned training length fails to converge. To address this issue, in our experiments, the training iterations of C-NCSN++ are reduced to 450,000 for the CIFAR-10 and ImageNet-32x32 datasets, 300,000 for the CIFAR-100 dataset, and 100,000 for the SVHN dataset. The training procedure of QC-NCSN++ consists of two stages. In the first stage, QC-NCSN++ is optimized in the same manner as U-NCSN++. In the second stage, the regularization term $\tilde{\mathcal{L}}_{\mathrm{QC}}^{\mathrm{est}}$ is incorporated during the training process, which requires additional 150,000 iterations for convergence. In the training process, the batch size b is fixed to 128 for C-NCSN++ and U-NCSN++, while its value is adjusted according to K for QC-NCSN++ for conserving the memory consumption. We adopt $(b, K) = (8, 16)$ in Table 2, and $(b, K) = (128, 1)$ in Tables 3 and 4 for QC-NCSN++, respectively. The maximal and minimal noise scales $\sigma_{\max}$ and $\sigma_{\min}$ are set to 50 and 0.01, respectively. The balancing factor λ is set to 0.0001. The ODE sampler is implemented using the `scipy.integrate.solve_ivp` library.

Evaluation Method. The asymmetry (*Asym*) and normalized asymmetry (*NAsym*) metrics are evaluated using Eqs. (A10) and (A11), respectively. They are formulated as follows:

$$\frac{1}{TMK}\sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{k=1}^{K}\mathbf{v}_k^T JJ^T\mathbf{v}_k - \mathbf{v}_k^T J J\mathbf{v}_k, \tag{A10}$$

$$\frac{1}{TMK}\sum_{t=1}^{T}\sum_{m=1}^{M}\sum_{k=1}^{K}\frac{\mathbf{v}_k^T JJ^T\mathbf{v}_k - \mathbf{v}_k^T J J\mathbf{v}_k}{\mathbf{v}_k^T JJ^T\mathbf{v}_k}, \tag{A11}$$

where $J = \frac{\partial}{\partial \tilde{\mathbf{x}}_m^{(t)}} s(\tilde{\mathbf{x}}_m^{(t)};\theta,\sigma_t)$, $\{\tilde{\mathbf{x}}_m^{(t)}\}_{m=1}^{M}$ represents a set of testing data points, and $\{\mathbf{v}_k\}_{k=1}^{K}$ is a set of i.i.d. samples drawn from $p_{\mathbf{v}}$. In our implementation, T and K are set as 100 and 1, respectively. The metrics for sampling performance (i.e., FID, IS, Precision and Recall) are evaluated using the `tensorflow_gan` library as well as the official evaluation package implemented by (Kynkäänniemi et al., 2019; Naeem et al., 2020).

Confidence Intervals of the Evaluation Results. Table A2 shows the 95% confidence intervals for the evaluation results of QC-NCSN++ in terms of the NLL, FID, IS, Precision, and Recall metrics on the CIFAR-10 dataset. For the evaluation results of the FID, IS, Precision, and Recall metrics, the PC sampler with NFE=1,000 is adopted. All of these results are obtained by three times of evaluations.

Table A2. The confidence interval of the evaluation results on the CIFAR-10 dataset.

NLL	FID	IS	Precision	Recall
±0.0014	±0.0143	±0.0065	±0.0024	±0.0011

Table A4. The time and memory consumption of evaluating the score function, the objective function, and the gradient of the objective function of C-NCSN++, U-NCSN++, and QC-NCSN++.

			Evaluating $\mathcal{L}(\theta)$		Evaluating $\frac{\partial}{\partial\theta}\mathcal{L}(\theta)$		Evaluating $s(\boldsymbol{x};\theta)$	
Method	Model ($s(\cdot\,;\theta)$)	Objective ($\mathcal{L}(\theta)$)	Time	Memory	Time	Memory	Time	Memory
C-NCSN++	Conservative NCSN++	$\mathcal{L}_{\mathrm{SM}}$	0.26 s	17.2 GB	1.21 s	19.3 GB	0.25 s	17.2 GB
U-NCSN++	NCSN++	$\mathcal{L}_{\mathrm{SM}}$	0.10 s	8.5 GB	0.30 s	8.9 GB	0.10 s	5.5 GB
QC-NCSN++	NCSN++	$\mathcal{L}_{\mathrm{Total}}$	0.36 s	27.6 GB	2.82 s	32.0 GB	0.10 s	5.5 GB

Sampling performance of U-NCSN++. Table A3 compares the sampling performance of the baseline method (i.e., U-NCSN++) reported in (Xu et al., 2022) and that reproduced by us on the CIFAR-10 dataset using an ODE sampler. It is observed that the reproduced results are improved in terms of the FID, IS, and NFE metrics. This reinforces our statement in Section 5, as QC-NCSN++ is able to achieve superior results to both the reproduced and reported performance of U-NCSN++.

Table A3. A comparison between the results reported in (Xu et al., 2022) and those reproduced by us for U-NCSN++.

	FID	IS	NFE
U-NCSN++ (Xu et al., 2022)	7.66	9.17	194
U-NCSN++ (Ours)	7.48	9.24	170
QC-NCSN++ (Ours)	**7.21**	**9.25**	**124**

A.7. Additional Experimental Results

In this section, we provide a number of additional experimental results. In Section A.7.1, we present additional experimental results of QCSBMs implemented as one-layered autoencoders to support our observation presented in Section 6 of the main manuscript. In Section A.7.2, we provide a comparison between C-NCSN++, U-NCSN++, and QC-NCSN++ in terms of their time and memory consumption for each training and sampling iteration. In Section A.7.3, we demonstrate the impact of the choices of λ on the performance of QC-NCSN++. Finally, in Section A.7.4, we provide additional qualitative results on the real-world datasets.

A.7.1. QCSBMs implemented as One-Layered Autoencoders

In Section 6, we leveraged the example of an one-layered autoencoder $s(\boldsymbol{x};\theta) = Rh(W^T\boldsymbol{x} + \boldsymbol{b}) + \boldsymbol{c}$ to demonstrate the advantage of QCSBMs over CSBMs. Our experimental results in Fig. 3 reveals that QCSBMs can learn to output conservative vector fields, which cannot be captured by CSBMs with tied weights (i.e., $R = W$). To further solidify our empirical observation, we provide additional examples in Fig. A4. Fig. A4 depicts the trends of $\left\|WR^T - RW^T\right\|_F$ and $\|W - R\|_F$ during the minimization process of $\mathcal{L}_{\mathrm{QC}}$ with four different seeds. As the training progresses, $\mathcal{L}_{\mathrm{QC}}$ and $\left\|WR^T - RW^T\right\|_F$ both approach to zero in all of these four examples. In contrast, the values of $\|W - R\|_F$ do not approach to zero, and the trends of $\|W - R\|_F$ for these four examples differ. The above experimental evidences demonstrate that QCSBMs can learn to output conservative vector fields with $R \neq W$, and thus justify the advantage of QCSBMs over CSBMs.

A.7.2. A Comparison on the Time and Memory Consumption

In this section, we investigate the time and memory consumption of evaluating the score function, the objective function, and the gradient of the objective function of C-NCSN++, U-NCSN++, and QC-NCSN++. The gradient operations with respect to both of the input $\boldsymbol{x}$ and the parameters θ are implemented using the automatic differentiation tool (Griewank & Walther, 2000) provided in the `Pytorch` library (Paszke et al., 2019). The results are evaluated on a single NVIDIA V100 GPU with 32 GB memory, and the batch size is fixed at 32. Table A4 reports the evaluation results of the above setting. It is observed that the training time and memory requirements of C-NCSN++ and QC-NCSN++ are higher than U-NCSN++ as the calculation of the objective $\mathcal{L}(\theta)$ and its gradients $\frac{\partial}{\partial\theta}\mathcal{L}(\theta)$ requires additional backward propagation. On the other hand, the sampling time and memory requirements of U-NCSN++ and QC-NCSN++ are lower than C-NCSN++, since the gradient operation in evaluating $s(\cdot\,;\theta)$ of C-NCSN++ is prevented.

A.7.3. The Impact of the Choices of λ on the Performance of QC-NCSN++

Based on our preliminary results on the toy environment, we perform a hyperparameter sweep for λ ={1e-3, 5e-4, 1e-4, 5e-5}, and report the best results on the real-world experiments. Table A5 presents the evaluation results of FID and IS under different choices of λ. In this experiment, the PC sampler is adopted and NFE is fixed at 1,000. The experimental results presented on the rows 'FID' and 'IS' demonstrate that QC-NCSN++ achieves its best sampling performance when λ is selected as 0.0001. Based on this finding, we choose λ to equal to 0.0001 throughout the experiments in Section 5.

Table A5. The evaluation results of QC-NCSN++ with different choices of λ on the CIFAR-10 dataset.

λ	0.001	0.0005	0.0001	0.00005	0.0
Asym	**9.99 e6**	2.38 e7	5.03 e7	6.42 e7	1.88 e8
FID	2.75	2.53	**2.48**	**2.48**	2.50
IS	9.58	9.64	**9.70**	9.61	9.58

A.7.4. Visualized Examples

Fig. A3 depict a few uncurated visualized examples that qualitatively demonstrate the sampling quality of QC-NCSN++ on the real-world datasets.

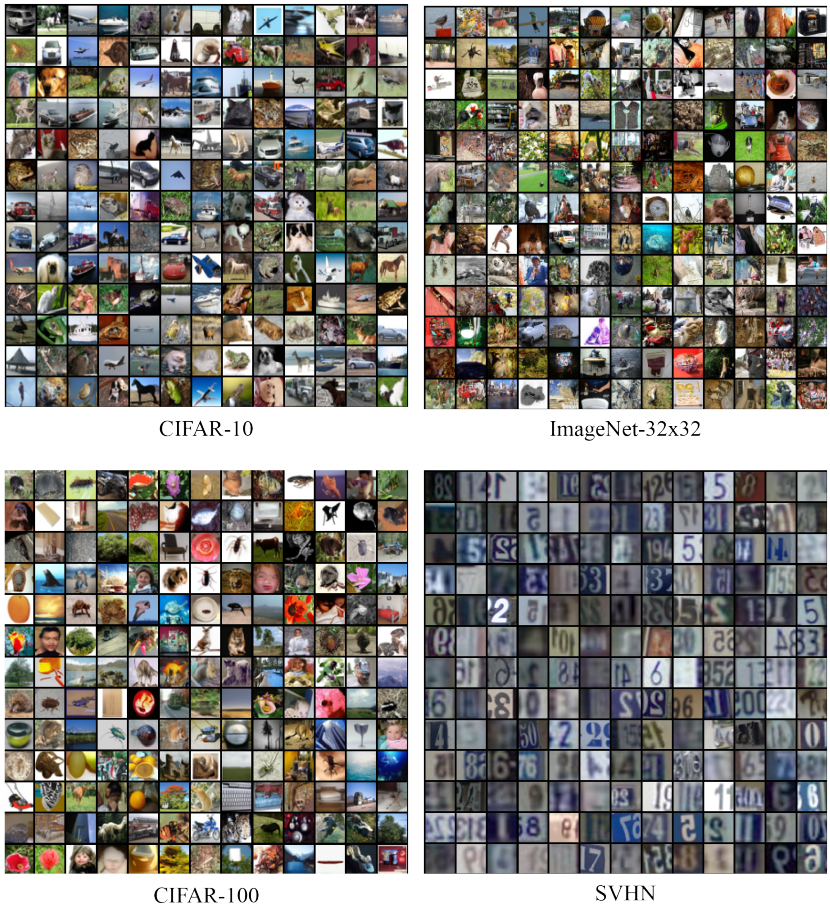


Figure A3. A number of visualized examples generated using QC-NCSN++.

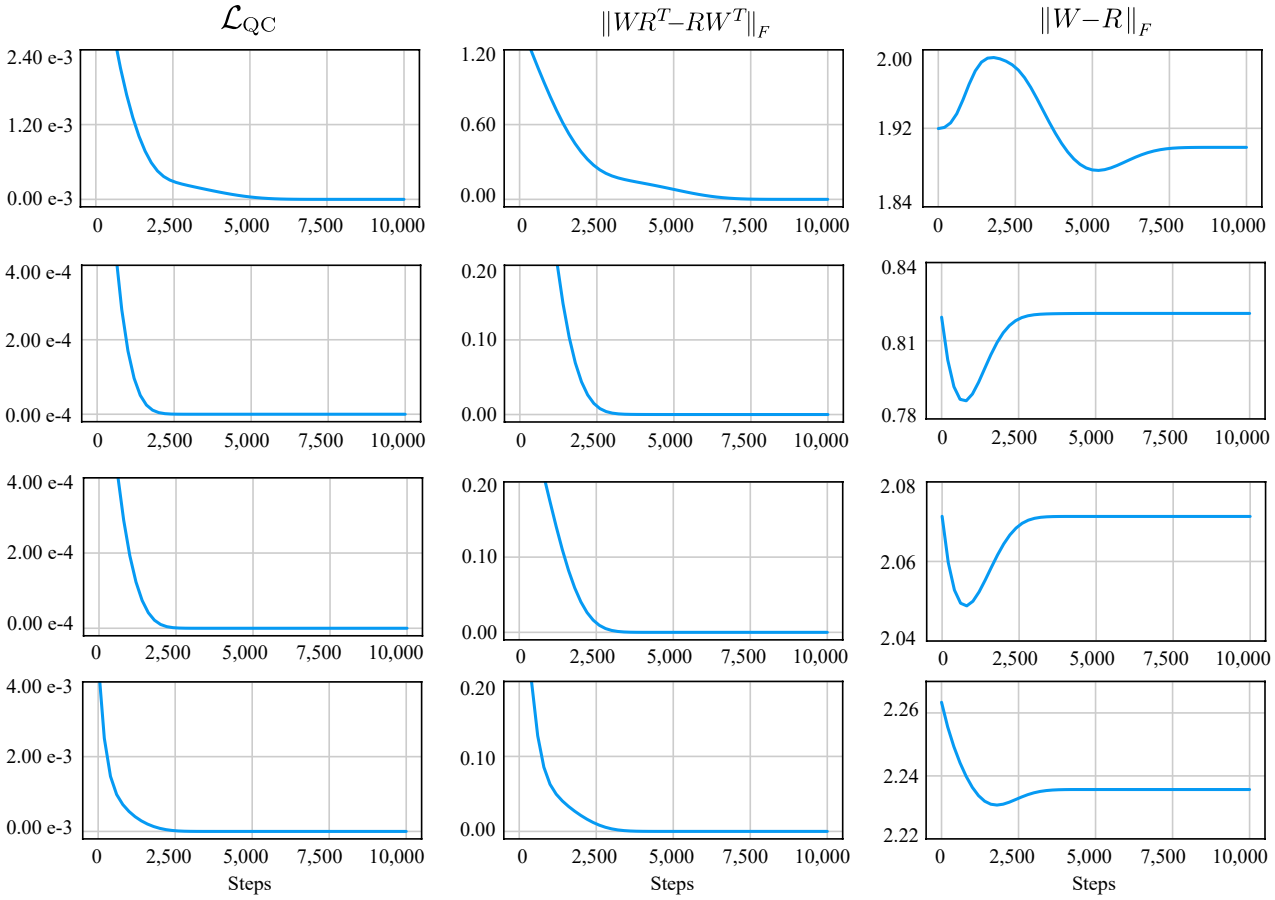


Figure A4. The trends of $\left\|WR^T - RW^T\right\|_F$ and $\|W - R\|_F$ during the minimization process of $\mathcal{L}_{\mathrm{QC}}$. The 'steps' on the x-axes refer to the training steps.