Understanding Bayesian Flow Networks

Jed Guzelkabaagac [1]

Abstract

Bayesian Flow Networks (BFNs) are a new class of generative models that combine Bayesian inference with neural networks. The network operates on the parameters of a data distribution, rather than on noisy data. The loss function directly optimizes data compression and imposes no constraints on network architecture. Experiments demonstrate competitive performance on image modeling tasks and superior results on the text8 character-level language modeling task, outperforming discrete diffusion models.

1. Introduction

Generative modeling has made significant strides in recent years, driven by neural networks' ability to capture complex relationships in high-dimensional data. Models like autoregressive networks (Sutskever et al., 2011), normalizing flows (Rezende & Mohamed, 2016), variational autoencoders (VAEs) (Vahdat & Kautz, 2021), and diffusion models (Sohl-Dickstein et al., 2015) achieve this by breaking down the joint distribution into smaller, manageable steps, mitigating the curse of dimensionality. These approaches are deeply tied to data compression, where minimizing the loss function aligns with efficient message transmission (Townsend et al., 2019). Autoregressive models excel at handling discrete data due to their sequential prediction framework, but they struggle with continuous data like images due to the lack of natural variable ordering and high computational cost for generation (Bond-Taylor et al., 2022). Diffusion models, in contrast, effectively generate images by progressively refining noisy representations (Rombach et al., 2022), but their application to discrete data is hindered by the discontinuous nature of discrete noise. Bayesian Flow Networks (Graves et al., 2024) address these challenges by operating on the parameters of data distributions instead of noisy data. This ensures a fully continuous and differentiable generative process, enabling gradient-based guidance and efficient generation across diverse domains.

[1]Technical University of Munich. Correspondence to: Jed Guzelkabaagac <jed.guzelkabaagac@tum.de>.

This paper provides an overview of Bayesian Flow Networks in Section 2, detailing their mechanisms for continuous data. Practical training considerations and loss functions for both discrete and continuous-time formulations are discussed in Section 3, followed by a critique and potential future directions in Section 4.

2. Bayesian Flow Networks

Bayesian Flow Networks (BFNs) are a novel class of generative models that learn distribution parameters directly, unlike diffusion models which operate on noisy data. BFNs utilize a **sender-receiver paradigm** where the sender distribution, parameterized by a known data point $\mathbf{x}$, transmits noisy observations during training. Observations are used to iteratively refine the parameters, $\boldsymbol{\theta}$, through Bayesian updates, as displayed in Figure 1. These parameters are then used as noisy inputs to a neural network, Ψ. The network produces a best guess $\hat{\mathbf{x}}$ of the true data, parameterizing the receiver distribution. This process progressively improves the receiver's estimate of the data distribution. The sender distribution acts as a "target" for the receiver distribution, implicitly encouraging the network Ψ to learn the underlying data distribution by maximizing the similarity between the sender (parameterized by true data $\mathbf{x}$) and the receiver (parameterized by the network's prediction $\hat{\mathbf{x}}$). Once trained, the receiver can generate novel samples independently, without access to the true data, by taking on the sampling role previously performed by the sender distribution. We can now delve into the specific roles of each distribution and the underlying mechanisms of a BFN.

2.1. Sender Distribution

The **sender distribution** at time t, denoted $p_S(\mathbf{y} \mid \mathbf{x}; \alpha_t)$, models the transmission of information about the true data $\mathbf{x}$ with an **accuracy parameter** $\alpha_t \in \mathbb{R}^+$. This parameter controls the informativeness of samples: at $\alpha_t = 0$, samples are uninformative, reflecting total uncertainty about $\mathbf{x}$. As α_t increases, the sender distribution concentrates around $\mathbf{x}$, becoming increasingly precise. The sender distribution is factorized as follows, where $\mathbf{y} = (y^{(1)}, \ldots, y^{(D)}) \in \mathcal{Y}^D$,

$$p_S(\mathbf{y} \mid \mathbf{x}; \alpha_t) = \prod_{d=1}^{D} p_S(y^{(d)} \mid x^{(d)}; \alpha_t).$$

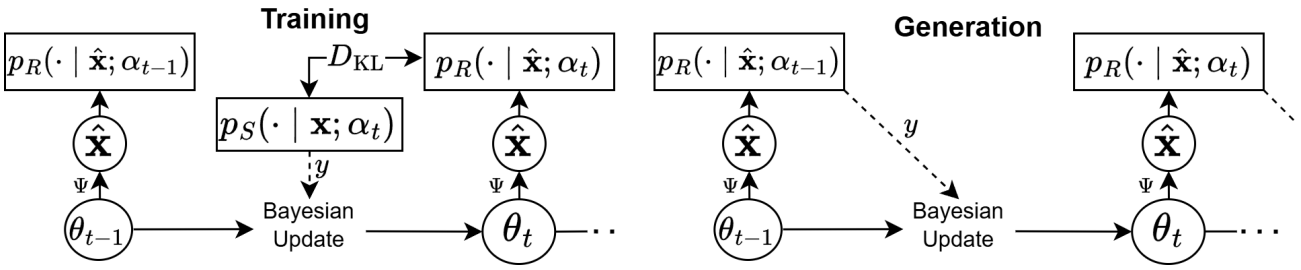

Figure 1. Simplified illustration of the training and generation processes in a BFN. For clarity, the steps from $\hat{x}$ to p_R are omitted. The figure highlights the roles of the sender and receiver distributions, as well as the Bayesian update, during both training and generation.

2.2. Bayesian Updates for Parameters

Operating on the parameter space is key to enabling a continuous and differentiable framework. We define the network parameters at time t as $\boldsymbol{\theta}_t = (\theta_t^{(1)}, \ldots, \theta_t^{(D)})$, where $\theta_t^{(d)}$ may represent probabilities in a categorical distribution, or the parameters of a Gaussian. During training, they are refined using Bayesian updates with samples $\mathbf{y}_t$ from the sender distribution, which contain further information about data point $\mathbf{x}$. Given the previous parameters $\boldsymbol{\theta}_{t-1}$, a noisy observation $\mathbf{y}_t$, and the accuracy parameter α_t, we deterministically compute the updated parameters $\boldsymbol{\theta}_t$ using an update rule denoted as $\boldsymbol{\theta}_t = h(\boldsymbol{\theta}_{t-1}, \mathbf{y}_t, \alpha_t)$. This update rule depends on the specific data type and distributions being modeled. Parameters $\boldsymbol{\theta}_t$ serve as guidance for the neural network during training, as they stochastically approach the true data $\mathbf{x}$ with "known accuracy". These parameters do not constitute the network's best estimate of $\mathbf{x}$, but rather serve as input for the network to learn at different "noise levels". Due to the factorization of the sender distribution, the individual components of the observations $\mathbf{y}_t$ are independent, leading to parameters $\boldsymbol{\theta}_t$ that are "naive" or uncontextualized during training. Each $\theta_t^{(i)}$ becomes "more sure" of the value $x^{(i)}$ as t increases, but without key contextual information about the other variables $x^{(j)}$.

The **update distribution**, denoted as $p_U(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_{t-1}, \mathbf{x}; \alpha_t)$, is obtained by marginalizing out observation $\mathbf{y}_t$ to directly model the evolution of the parameters $\boldsymbol{\theta}_t$ over t. We want p_U to possess the attractive property of **additive accuracies**:

$$\mathop{\mathbb{E}}_{p_U(\boldsymbol{\theta}' \mid \boldsymbol{\theta}, \mathbf{x}; \alpha_a)} p_U(\boldsymbol{\theta}'' \mid \boldsymbol{\theta}', \mathbf{x}; \alpha_b) = p_U(\boldsymbol{\theta}'' \mid \boldsymbol{\theta}, \mathbf{x}; \alpha_a + \alpha_b).$$

Intuitively, if we marginalize over an intermediate parameter $\boldsymbol{\theta}'$, the distribution of the final parameter $\boldsymbol{\theta}''$ is the same as if we had directly evolved $\boldsymbol{\theta}$ from the initial step, with the accuracies α_a and α_b simply being added. Crucially, this allows us to sample $\boldsymbol{\theta}_t$ directly from the prior $\boldsymbol{\theta}_0$ using the known distribution p_U. In turn, this update distribution p_U is pivotal for defining the *n*-step loss function (as discussed in Section 3), and also underpins the derivation of the Bayesian flow distribution needed for the time-continuous loss.

2.3. Output and Receiver Distributions

The parameters $\boldsymbol{\theta}_{t-1}$ and the process time $t-1$ are passed as input to a neural network, denoted by Ψ. The network outputs the best guess of $\mathbf{x}$ at time t, denoted

$$\hat{\mathbf{x}}_t \triangleq \Psi(\boldsymbol{\theta}_{t-1}, t-1).$$

This parameterizes the **output distribution** at time t, written as $p_O(\mathbf{x} \mid \hat{\mathbf{x}}_t)$. For example, if $\mathbf{x}$ is continuous, the output distribution at time t would simply be a Dirac delta distribution centered at $\hat{\mathbf{x}}_t$.

A key distinction arises in the expressiveness of the parameters $\boldsymbol{\theta}_t$ and the output of the network. While the parameters are "naive" due to the factorization of the sender distribution, the output distribution leverages the neural network to "join" these independent components into a single, contextualized representation, reflecting relationships across all elements of $\boldsymbol{\theta}_t$. This enables the output distribution to exploit dependencies among data components, such as neighboring pixels in an image or related words in a text.

The **receiver distribution** at time t, denoted as p_R, is defined as the expectation of the sender distribution over the output distribution:

$$p_R(\mathbf{y} \mid \hat{\mathbf{x}}_t; \alpha_t) \triangleq \mathop{\mathbb{E}}_{p_O(\mathbf{x}' \mid \hat{\mathbf{x}}_t)} p_S(\mathbf{y} \mid \mathbf{x}'; \alpha_t). \tag{1}$$

Intuitively, the receiver distribution represents the average of all possible sender distributions, weighted by the current belief about the data, which is characterized by the output distribution. This averaging effectively combines the sender's inherent uncertainty (quantified by α_t) with the receiver's uncertainty about the true data, allowing the receiver to refine its overall belief over time. Furthermore, if the output distribution places more density at the true value $\mathbf{x}$, the receiver distribution will more closely resemble the sender distribution, which is parameterized by $\mathbf{x}$. This connection underscores the need to minimize the KL divergence between the sender and receiver distributions to drive accurate estimates $\hat{\mathbf{x}}_t$ of the true data. In Section 3, we elaborate on how the discrete- and continuous-time losses achieve this.

2.4. Example: Continuous Data

We demonstrate the application of BFNs to continuous data. Network parameters at time t are $\boldsymbol{\theta}_t = \boldsymbol{\mu}_t$. We choose an accuracy schedule $\alpha_1, ..., \alpha_n$ and define $\rho_t = \rho_0 + \sum_{k=1}^{t} \alpha_k$. Initialize parameters $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\rho_0 = 1$. The sender distribution at time t, with precision controlled by α_t, is given by:

$$p_S(\mathbf{y} \mid \mathbf{x}; \alpha_t) = \mathcal{N}(\mathbf{y} \mid \mathbf{x}, \alpha_t^{-1}\mathbf{I}). \tag{2}$$

This distribution is relevant during training, where we have access to the true data $\mathbf{x}$. At each time step, the parameters $\boldsymbol{\mu}_t$ are refined using Bayesian updates with samples $\mathbf{y}_t$ from the sender or receiver distribution. The Bayesian update function h for the mean is given by:

$$\boldsymbol{\mu}_t = h(\boldsymbol{\mu}_{t-1}, \mathbf{y}_t, \alpha_t) = \frac{\alpha_t \mathbf{y}_t + \rho_{t-1}\boldsymbol{\mu}_{t-1}}{\rho_t}. \tag{3}$$

At time $t-1$, parameters $\boldsymbol{\mu}_{t-1}$ and the process time are passed as input to a neural network, denoted by Ψ. The network outputs our current best guess of $\mathbf{x}$, denoted $\hat{\mathbf{x}}_t \triangleq \Psi(\boldsymbol{\mu}_{t-1}, t-1)$. This estimate parameterizes the **output distribution** at time t, defined as a Dirac delta function centered at $\hat{\mathbf{x}}$:

$$p_O(\mathbf{x} \mid \hat{\mathbf{x}}_t) = \delta(\mathbf{x} - \hat{\mathbf{x}}_t). \tag{4}$$

The **receiver distribution** at time t represents a noisy version of the model's current belief about the data. It is derived directly from Equations (2) and (4), as per Equation (1):

$$\begin{aligned} p_R(\mathbf{y} \mid \hat{\mathbf{x}}_t; \alpha_t) &= \mathop{\mathbb{E}}_{\delta(\mathbf{x}' - \hat{\mathbf{x}}_t)} \mathcal{N}(\mathbf{y} \mid \mathbf{x}', \alpha_t^{-1}\mathbf{I}) \\ &= \mathcal{N}(\mathbf{y} \mid \hat{\mathbf{x}}_t, \alpha_t^{-1}\mathbf{I}). \end{aligned}$$

While directly comparing $\mathbf{x}$ and $\hat{\mathbf{x}}$ may suffice in a continuous setting, this noisy formulation—focusing on the sender and receiver distributions as in Figure 2—is crucial for a unified framework. It ensures continuity, differentiability, and applicability to diverse data types, such as discrete and discretized data, where the output distribution may not be a Dirac delta.

By marginalizing out observation $\mathbf{y}_t$ from Equation (3), we can directly find the evolution of parameter $\boldsymbol{\mu}_t$:

$$p_U(\boldsymbol{\mu}_t \mid \boldsymbol{\mu}_{t-1}, \mathbf{x}; \alpha_t) = \mathcal{N}\left(\boldsymbol{\mu}_t \mid \frac{\alpha_t \mathbf{x} + \boldsymbol{\mu}_{t-1}\rho_{t-1}}{\rho_t}, \frac{\alpha_t}{\rho_t^2}\mathbf{I}\right).$$

After verifying that the additive accuracies property holds, we can derive the distribution of $\boldsymbol{\mu}_t$ directly from the initialization, known data point $\mathbf{x}$, and $\rho_t = 1 + \sum_{i=1}^{t} \alpha_i$:

$$p_U(\boldsymbol{\mu}_t \mid \boldsymbol{\mu}_0, \mathbf{x}; \rho_t - 1) = \mathcal{N}\left(\boldsymbol{\mu}_t \mid \frac{\rho_t - 1}{\rho_t}\mathbf{x}, \frac{\rho_t - 1}{\rho_t^2}\mathbf{I}\right).$$

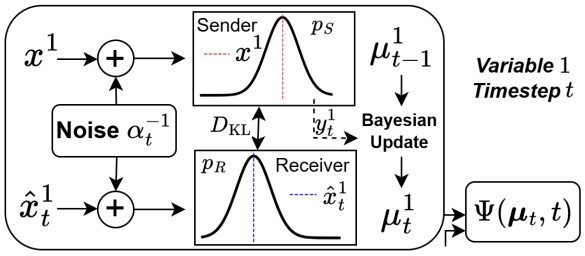

Figure 2. Training step of a single variable in a continuous BFN.

3. Training and Sample Generation

The loss function guides the neural network Ψ to learn the underlying data distribution. Similar to diffusion models, this is achieved by minimizing a KL divergence.

3.1. Accuracy Schedule and Bayesian Flow Distribution

Let $t \in [0, 1]$ be the normalized time, and $\alpha(t) > 0$ be the accuracy rate at time t. The **accuracy schedule** $\beta(t)$ is defined as:

$$\beta(t) = \int_0^t \alpha(t')\, dt'.$$

Intuitively, the accuracy schedule $\beta(t)$ represents the cumulative information gained about the true data $\mathbf{x}$ up to time t. It is the natural, continuous time extension of the accumulated accuracy parameter ρ_t in discrete time. The **Bayesian flow distribution** provides the distribution of the parameters $\boldsymbol{\theta}$ at *any* time t, and can be viewed as a continuous extension of the update distribution:

$$\begin{aligned} p_F(\boldsymbol{\theta} \mid \mathbf{x}; t) &\triangleq p_U(\boldsymbol{\theta} \mid \boldsymbol{\theta}_0, \mathbf{x}; \beta(t)) \\ &= \mathcal{N}\left(\boldsymbol{\mu} \mid \frac{\beta(t)}{1+\beta(t)}\mathbf{x}, \frac{\beta(t)}{(1+\beta(t))^2} I\right), \end{aligned}$$

where the second equality holds for the continuous-data case. Denoting σ_1 as the standard deviation of $\boldsymbol{\mu}_1$, we set $\beta(t) \triangleq \sigma_1^{-2t} - 1$ so that the expected entropy of the input distribution decreases linearly with t.

In practice, $p_F(\boldsymbol{\theta} \mid \mathbf{x}; t)$ greatly simplifies training by eliminating the need to handle the entire parameter chain as seen in Equation (5).

3.2. Discrete and Continuous Time Loss

The discrete-time loss, denoted as $L^n(\mathbf{x})$ in Equation (5), is conceptually similar to the negative variational lower bound in a VAE, where the noisy observations act as latent variables. Minimizing the loss corresponds to maximizing a lower bound on the log-likelihood of the data. However, directly computing the expectation in Equation (5) involves integrating over the joint distribution of parameters at each time step, which is computationally expensive. Instead, we leverage the Bayesian flow distribution, resulting in Equation (6), allowing for efficient Monte Carlo sampling.

$$L^n(\mathbf{x}) \triangleq \mathbb{E}_{p(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_{n-1})} \sum_{i=1}^{n} D_{KL}\left(p_S(\cdot \mid \mathbf{x};\alpha_i) \,\|\, p_R(\cdot \mid \hat{\mathbf{x}}(\boldsymbol{\theta}_{i-1}, i-1);\alpha_i)\right) \tag{5}$$

$$L^n(\mathbf{x}) = n\mathbb{E}_{i\sim U(1,n),\, p_F(\theta|\mathbf{x},t_{i-1})} D_{KL}\left(p_S(\cdot \mid \mathbf{x};\alpha_i) \,\|\, p_R(\cdot \mid \hat{\mathbf{x}}(\boldsymbol{\theta}, i-1);\alpha_i)\right) \tag{6}$$

$$L^\infty(\mathbf{x}) = \lim_{\epsilon\to 0} \frac{1}{\epsilon} \mathbb{E}_{t\sim U(\epsilon,1),\, p_F(\theta|\mathbf{x},t-\epsilon)} D_{KL}\left(p_S(\cdot \mid \mathbf{x};\alpha(t,\epsilon)) \,\|\, p_R(\cdot \mid \hat{\mathbf{x}}(\boldsymbol{\theta}, t-\epsilon);\alpha(t,\epsilon))\right) \tag{7}$$

Table 1. Loss functions for BFNs. We utilize the explicit notation $\hat{\mathbf{x}}(\boldsymbol{\theta}, t) \triangleq \Psi(\boldsymbol{\theta}, t)$ to unify the notations across equations. In continuous time this can be viewed as a 'current estimate' known at t, as timesteps are infinitesimally small. Here, $\alpha(t,\epsilon) \triangleq \beta(t) - \beta(t-\epsilon)$ represents the accuracy gained over the infinitesimal interval from $t-\epsilon$ to t.

While $L^n(\mathbf{x})$ can be used for training, it has two main drawbacks. Firstly, it requires fixing the number of timesteps, n, during training. Secondly, for discrete or discretized data, the KL terms do not have analytic solutions, leading to noisy gradient estimates. To address these issues, we derive a continuous-time loss function, $L^\infty(x)$, by taking the limit of $L^n(x)$ as $n \to \infty$, drawing inspiration from Variational Diffusion Models (Kingma et al., 2023). This continuous-time loss, shown in Equation (7), is mathematically simpler and eliminates the need to fix n during training. Pseudo-code in the continuous data case is shown in Algorithm 1, where the final line comes from evaluating the KL divergence. Note that the pseudo-code is adapted from the original paper to reflect notation changes.

Algorithm 1 Continuous-Time Loss $L^\infty(\mathbf{x})$: Cts. Data

Require: $\sigma_1 \in \mathbb{R}^+$
input Continuous data $\mathbf{x} \in \mathbb{R}^D$
$t \sim U(0,1)$
$\gamma \leftarrow 1 - \sigma_1^{2t}$
$\boldsymbol{\mu} \sim \mathcal{N}(\gamma\mathbf{x}, \gamma(1-\gamma)I)$
$\hat{\mathbf{x}}(\boldsymbol{\theta}, t) \leftarrow \Psi(\boldsymbol{\mu}, t)$
$L^\infty(\mathbf{x}) \leftarrow -\ln\sigma_1 \sigma_1^{-2t} \cdot \|\mathbf{x} - \hat{\mathbf{x}}(\boldsymbol{\theta}, t)\|^2$

3.3. Sample Generation

While training utilizes a continuous-time loss function, the generation process involves a discrete-time sampling procedure. Starting with an initialization of $\boldsymbol{\mu}$ and ρ, we iteratively refine our estimate using the learned network Ψ as follows.

Algorithm 2 Sample Generation: Cts. Data

Require: $\sigma_1 \in \mathbb{R}^+$, number of steps $n \in \mathbb{N}$
$\boldsymbol{\mu} \leftarrow 0$, $\rho \leftarrow 1$
for $i = 1$ **to** n **do**
  $t \leftarrow \frac{i-1}{n}$
  $\hat{\mathbf{x}}(\boldsymbol{\theta}, t) \leftarrow \Psi(\boldsymbol{\mu}, t)$
  $\alpha \leftarrow \sigma^{-2i/n} \frac{1}{1-\sigma^{2/n}}$
  $\mathbf{y} \sim \mathcal{N}\left(\hat{\mathbf{x}}(\boldsymbol{\theta}, t), \alpha^{-1}I\right)$
  $\boldsymbol{\mu} \leftarrow \frac{\rho\boldsymbol{\mu} + \alpha\mathbf{y}}{\rho+\alpha}$, $\rho \leftarrow \rho + \alpha$
end for
$\hat{\mathbf{x}}(\boldsymbol{\theta}, 1) \leftarrow \Psi(\boldsymbol{\mu}, 1)$

4. Discussion

BFNs have demonstrated competitive performance on tasks like language modeling, particularly excelling with discrete data where they outperform discrete diffusion models. Despite their promise, BFNs present certain challenges and open questions that necessitate further investigation. One of the primary hurdles is in their computational cost, especially during training. This necessitates research into more efficient training methods to make BFNs more accessible for wider use. Furthermore, the selection of the accuracy schedule is a critical component which requires careful consideration as the optimal choice can significantly impact performance and varies across applications. Developing principled approaches for selecting or adapting this schedule is crucial for improving BFNs' efficacy and generalization. While BFNs possess a strong theoretical foundation, they are a relatively new class of generative models, and certain limitations and open questions remain. For example, while BFNs have shown promising results on dynamically binarized MNIST, which tests a model's ability to handle variability and learn complex distributions, their performance on a wider range of image generation tasks requires further investigation.

4.1. Applications

BFNs have already found successful applications in diverse domains. Beyond general use cases like image generation, language modeling, and time series prediction, BFNs have been extended to address specific challenges in specialized areas. GeoBFNs have been developed for 3D molecule generation, overcoming issues of multi-modality and noise sensitivity to achieve state-of-the-art performance (Song et al., 2024). Similarly, MolJO employs BFNs for structure-based molecule optimization, effectively handling the joint optimization of continuous and discrete properties (Qiu et al., 2024). Furthermore, Marked Temporal BFN Point Processes (BMTPP) have shown promise in modeling complex event sequences with timestamps and types (Chen et al., 2024). While BFNs have achieved state-of-the-art performance on such tasks, further comparison with other emerging methods, such as Discrete Flow Models (Campbell et al., 2024), is crucial to gain a comprehensive understanding of the relative strengths and weaknesses of each approach.

References

Bond-Taylor, S., Leach, A., Long, Y., and Willcocks, C. G. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, 2022. doi: 10.1109/TPAMI.2021.3116668.

Campbell, A., Yim, J., Barzilay, R., Rainforth, T., and Jaakkola, T. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design, 2024. URL https://arxiv.org/abs/2402.04997.

Chen, H., Fan, X., Liu, H., and Cao, L. Marked temporal bayesian flow point processes, 2024. URL https://arxiv.org/abs/2410.19512.

Graves, A., Srivastava, R. K., Atkinson, T., and Gomez, F. Bayesian flow networks, 2024. URL https://arxiv.org/abs/2308.07037.

Kingma, D. P., Salimans, T., Poole, B., and Ho, J. Variational diffusion models, 2023. URL https://arxiv.org/abs/2107.00630.

Qiu, K., Song, Y., Yu, J., Ma, H., Cao, Z., Zhang, Z., Wu, Y., Zheng, M., Zhou, H., and Ma, W.-Y. Structure-based molecule optimization via gradient-guided bayesian update, 2024. URL https://arxiv.org/abs/2411.13280.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows, 2016. URL https://arxiv.org/abs/1505.05770.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models, 2022. URL https://arxiv.org/abs/2112.10752.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. URL https://arxiv.org/abs/1503.03585.

Song, Y., Gong, J., Qu, Y., Zhou, H., Zheng, M., Liu, J., and Ma, W.-Y. Unified generative modeling of 3d molecules via bayesian flow networks, 2024. URL https://arxiv.org/abs/2403.15441.

Sutskever, I., Martens, J., and Hinton, G. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pp. 1017–1024, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Townsend, J., Bird, T., and Barber, D. Practical lossless compression with latent variables using bits back coding, 2019. URL https://arxiv.org/abs/1901.04866.

Vahdat, A. and Kautz, J. Nvae: A deep hierarchical variational autoencoder, 2021. URL https://arxiv.org/abs/2007.03898.