Conditional Bayesian Flow Networks

Yishen Li[1,5] Yunze Wu[1,2,4,5,*] Jiaming Liu[1,2,5]
Yuhang He[1,3,5]
School of Mathematical Sciences, Peking University
{2200010846, 2200010792, 2200010893, 2200010881}@stu.pku.edu.cn

Abstract

Bayesian Flow Networks (BFNs) (1) are a class of innovative generative models. They model data by iteratively updating the parameters of factorized input distributions via Bayesian inference on noisy data samples. This process is guided by a neural network that takes the current input distribution parameters and contextual information (e.g., time) and outputs the parameters for a secondary output distribution. Unlike diffusion models, BFNs operate directly on distribution parameters without defining a forward process, allowing them to maintain continuity and differentiability even when handling discrete data. Based on the original BFN framework, this paper focuses on studying and extending its conditional generative capabilities. We introduce a conditioning variable c as an additional input to the neural network and accordingly modify the key components of the model, including the output distribution, receiver distribution, and loss function. This extension enables BFNs to precisely guide and control the generation process based on external information (e.g., class labels). We detail the mathematical framework of conditional BFNs, derive their discrete- and continuous-time loss functions, and present the corresponding conditional sampling algorithm. Preliminary experimental results on the dynamically Binarized MNIST dataset show that our proposed Conditional BFNs can effectively generate image samples corresponding to specified class labels, demonstrating the potential of this framework in controllable generation tasks. Our code is available at `https://github.com/wyzmike05/2025BayesianFinalProject`.

1 Introduction

In recent years, large-scale neural networks have achieved revolutionary progress in the field of generative modeling, demonstrating exceptional capabilities in capturing complex dependencies within high-dimensional data. Models such as autoregressive models (2), flow-based models (3), deep variational autoencoders (VAEs) (4), and diffusion models (5) all share a core advantage: the ability to decompose complex joint distributions into a sequence of more tractable steps, thus effectively circumventing the "curse of dimensionality" .

*Corresponding author.

[1]Theory: Yishen Li (derivation reproduction), Yunze Wu (conditional generation), Jiaming Liu, Yuhang He. We read and analyzed the content of the original paper together.

[2]Implementation and Experiments: Jiaming Liu, Yunze Wu.

[3]Writing: Yuhang He (main).

[4]Presentations: Yunze Wu (midterm & final).

[5]Slides: Yuhang He (midterm); Yishen Li, Jiaming Liu, Yunze Wu (final).

Preprint. Under review.

Bayesian Flow Networks (BFNs) (1), as an emerging generative modeling framework, offer a unique perspective to understand and construct such stepwise refinement generative processes. The core idea can be interpreted via an information-theoretic metaphor: a sender, Alice, intends to efficiently transmit data to a receiver, Bob. In BFNs, this process is modeled as Bob holding an 'input distribution' (initially a simple prior) and, at each transmission step, feeding its parameters (for example, the mean of a Gaussian or the probabilities of a categorical distribution) into a neural network. The network then outputs the parameters of an "output distribution" . Meanwhile, Alice creates a "sender distribution" by adding noise to the real data, and Bob constructs a "receiver distribution" by convolving the output distribution with the same noise. Alice sends a sample from the sender distribution, and its transmission cost (i.e., its contribution to the loss function) equals the KL divergence from the sender to the receiver distribution. Bob uses the received sample to update the parameters of his input distribution through Bayesian inference. This iterative process is repeated, progressively refining Bob's estimate of the data.

Compared with widely used diffusion models, a distinctive feature of BFNs is that the network directly operates on the parameters of the data distribution, rather than on a noise-corrupted version of the data. This ensures that the entire generative process remains continuous and differentiable even when handling discrete data, providing a theoretical foundation for gradient-based guidance and few-step generation in discrete domains.

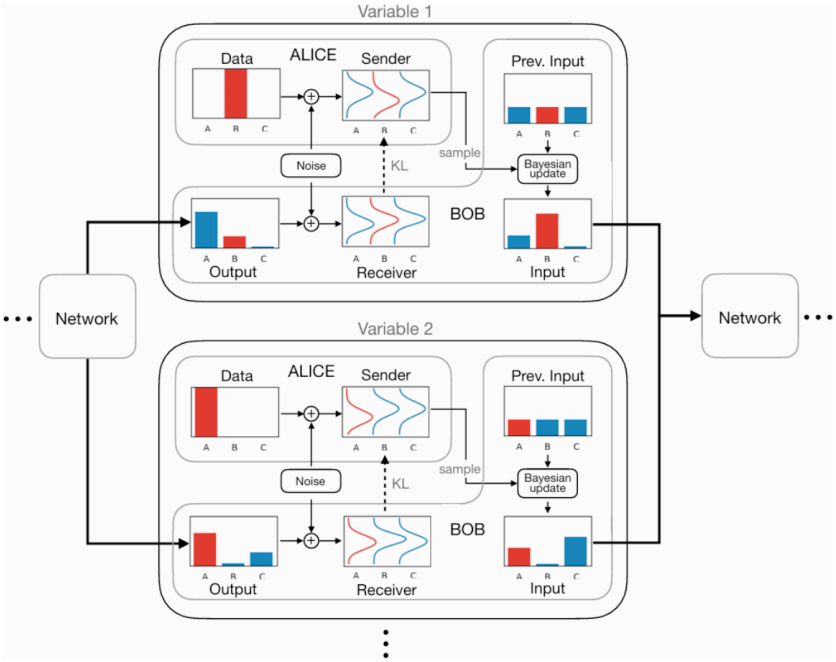


Figure 1: System overview of Bayesian Flow Networks (1). The figure illustrates one step in the modeling process of a BFN. The data, shown here as a sequence of ternary symbols, displays the first two variables (e.g., 'B' and 'A'). At each step, the network emits the parameters of the output distribution based on the parameters of the previous input distribution (and, in our conditional extension, a condition c). The sender and receiver distributions (both continuous even if the data is discrete) are constructed by adding random noise to the data and the output distribution, respectively. A sample from the sender distribution is then used to update the parameters of the input distribution, following the rules of Bayesian inference. Conceptually, this is a message sent from Alice to Bob, whose contribution to the loss function is the KL divergence from the sender to the receiver distribution.

However, the original BFN framework primarily focuses on unconditional generation, i.e., generating data from a prior distribution, lacking the ability to guide the generation process using specific external information. In many practical applications—such as generating

object images based on class labels, synthesizing scenes from text descriptions, or designing molecules with desired properties—controllable conditional generation is of critical importance.

The main contribution of this paper is to extend the original BFN framework to conditional generation scenarios by proposing Conditional Bayesian Flow Networks (Conditional BFNs). We introduce a conditioning variable $c \in \mathcal{C}$ as an additional input to the neural network at each time step, enabling the network to dynamically adjust the parameters of the output distribution based on c. Accordingly, the receiver distribution and the model's loss function are also adjusted to incorporate the conditioning. This design allows BFNs to effectively leverage external information (e.g., class labels, text embeddings, or other modality data) to precisely guide and control the direction and content of data generation.

In this work, we provide a detailed derivation of the mathematical formalism of Conditional BFNs, including the conditioned output distribution, receiver distribution, discrete-time and continuous-time loss functions, and the corresponding conditional sampling algorithm. Furthermore, we conduct preliminary experiments on the dynamically Binarized MNIST dataset to demonstrate the capability of Conditional BFNs to generate digit images corresponding to specified class labels. We also provide a visualization analysis of the evolution of Bayesian flows under conditioning to validate the effectiveness of our proposed conditioning mechanism. This work aims to lay the foundation for applying BFNs to controllable generation tasks and offers a new perspective for exploring more general conditional generation scenarios.

1.1 Input Distribution and Sender Distribution

Given D-dimensional data $\mathbf{x} = (x^{(1)}, \ldots, x^{(D)}) \in \mathcal{X}^D$ and a conditioning variable $c \in \mathcal{C}$, let $\boldsymbol{\theta} = (\theta^{(1)}, \ldots, \theta^{(D)})$ denote the parameters of a factorized input distribution $p_I(\cdot|\boldsymbol{\theta})$, where:

$$p_I(\mathbf{x}|\boldsymbol{\theta}) = \prod_{d=1}^{D} p_I(x^{(d)}|\theta^{(d)}) \tag{1}$$

For example, $\theta^{(d)}$ may encode the probabilities of a categorical distribution.

Let $p_S(\cdot|\mathbf{x};\alpha)$ denote a similarly factorized sender distribution, where $\mathbf{y} = (y^{(1)}, \ldots, y^{(D)}) \in \mathcal{Y}^D$ and:

$$p_S(\mathbf{y}|\mathbf{x};\alpha) = \prod_{d=1}^{D} p_S(y^{(d)}|x^{(d)};\alpha) \tag{2}$$

Here, $\alpha \in \mathbb{R}^+$ is an accuracy parameter, defined such that when $\alpha = 0$, the sender samples contain no information about $\mathbf{x}$, and as α increases, the samples become increasingly informative.

1.2 Output Distribution $p_O(\cdot|\boldsymbol{\theta}, c, t)$

During the data transmission process, the input parameters $\boldsymbol{\theta}$, the conditioning variable c, and the processing time t are provided as inputs to a neural network Ψ. The network then outputs a vector $\Psi(\boldsymbol{\theta}, c, t) = (\Psi^{(1)}(\boldsymbol{\theta}, c, t), \ldots, \Psi^{(D)}(\boldsymbol{\theta}, c, t))$, which parameterizes an output distribution with the same factorization structure as the input and sender distributions:

$$p_O(\mathbf{x}|\boldsymbol{\theta}, c, t) = \prod_{d=1}^{D} p_O(x^{(d)}|\Psi^{(d)}(\boldsymbol{\theta}, c, t)) \tag{3}$$

The introduction of the conditioning variable c is central to this framework, as it allows the network to adjust its output according to c, thereby enabling conditional generation.

1.3 Receiver Distribution $p_R(\cdot|\boldsymbol{\theta}; t, c, \alpha)$

Given the sender distribution $p_S(\cdot|\mathbf{x};\alpha)$ and the conditional output distribution $p_O(\cdot|\boldsymbol{\theta}, c, t)$, the receiver distribution defined over $\mathcal{Y}^D$ is:

$$p_R(\mathbf{y}|\boldsymbol{\theta}; t, c, \alpha) = \mathbb{E}_{p_O(\mathbf{x}'|\boldsymbol{\theta}, c, t)}[p_S(\mathbf{y}|\mathbf{x}';\alpha)] \tag{4}$$

Intuitively, this can be understood as a receiver who knows the functional form of the sender distribution $p_S(\cdot|\mathbf{x};\alpha)$ but does not know $\mathbf{x}$. Therefore, it integrates over all possible $\mathbf{x}' \in \mathcal{X}^D$ (i.e., all potential sender distributions), with each $\mathbf{x}'$ weighted by its probability under the conditional output distribution $p_O(\mathbf{x}'|\boldsymbol{\theta}, c, t)$.

1.4 Bayesian Update

Given parameters $\boldsymbol{\theta}$ and a sender sample $\mathbf{y}$ drawn at accuracy α, the Bayesian update function h computes the updated parameters $\boldsymbol{\theta}'$ by applying the rule of Bayesian inference:

$$\boldsymbol{\theta}' \leftarrow h(\boldsymbol{\theta}, \mathbf{y}, \alpha) \tag{5}$$

The Bayesian update distribution $p_U(\cdot|\boldsymbol{\theta}, \mathbf{x}; \alpha)$ is then defined by marginalizing over $\mathbf{y}$:

$$p_U(\boldsymbol{\theta}'|\boldsymbol{\theta}, \mathbf{x}; \alpha) = \mathbb{E}_{p_S(\mathbf{y}|\mathbf{x};\alpha)}[\delta(\boldsymbol{\theta}' - h(\boldsymbol{\theta}, \mathbf{y}, \alpha))] \tag{6}$$

where $\delta(\cdot - \mathbf{a})$ denotes the multivariate Dirac delta distribution centered at the vector $\mathbf{a}$. As shown in the original work, the form of $p_U(\cdot|\boldsymbol{\theta}, \mathbf{x}; \alpha)$ considered here exhibits accuracy additivity: if $\alpha = \alpha_a + \alpha_b$, then

$$p_U(\boldsymbol{\theta}''|\boldsymbol{\theta}, \mathbf{x}; \alpha) = \mathbb{E}_{p_U(\boldsymbol{\theta}'|\boldsymbol{\theta},\mathbf{x};\alpha_a)}[p_U(\boldsymbol{\theta}''|\boldsymbol{\theta}', \mathbf{x}; \alpha_b)] \tag{7}$$

From this property, it follows that given prior input parameters $\boldsymbol{\theta}_0$, the probability of observing $\boldsymbol{\theta}_n$ after drawing a sequence of n sender samples $\mathbf{y}_1, \ldots, \mathbf{y}_n$ at respective accuracies $\alpha_1, \ldots, \alpha_n$ is:

$$\mathbb{E}_{p_U(\boldsymbol{\theta}_1|\boldsymbol{\theta}_0,\mathbf{x};\alpha_1)} \cdots \mathbb{E}_{p_U(\boldsymbol{\theta}_{n-1}|\boldsymbol{\theta}_{n-2},\mathbf{x};\alpha_{n-1})} p_U(\boldsymbol{\theta}_n|\boldsymbol{\theta}_{n-1}, \mathbf{x}; \alpha_n) = p_U\left(\boldsymbol{\theta}_n \mid \boldsymbol{\theta}_0, \mathbf{x}; \sum_{i=1}^{n} \alpha_i\right) \tag{8}$$

1.5 Accuracy Schedule $\beta(t)$

By performing an infinite number of transmission steps, the Bayesian update process can be extended to continuous time. Let $t \in [0, 1]$ denote process time, and let $\alpha(t) > 0$ be the accuracy rate at time t. The accuracy schedule $\beta(t)$ is then defined as:

$$\beta(t) = \int_0^t \alpha(t')dt' \tag{9}$$

By definition, $\beta(t)$ is a monotonically increasing function of t, with $\beta(0) = 0$, and $\frac{d\beta(t)}{dt} = \alpha(t)$.

1.6 Bayesian Flow Distribution $p_F(\cdot|\mathbf{x}; t)$

Given prior parameters $\boldsymbol{\theta}_0$, the Bayesian update distribution $p_U(\cdot|\boldsymbol{\theta}, \mathbf{x}; \alpha)$, and the accuracy schedule $\beta(t)$, the Bayesian flow distribution $p_F(\cdot|\mathbf{x}; t)$ is defined as the marginal distribution of the input parameters at time t:

$$p_F(\boldsymbol{\theta}|\mathbf{x}; t) = p_U(\boldsymbol{\theta}|\boldsymbol{\theta}_0, \mathbf{x}; \beta(t)) \tag{10}$$

Note that the definition of the Bayesian flow distribution p_F does not explicitly depend on the condition c. It describes how the parameters evolve given the true data $\mathbf{x}$. The condition c influences the overall generative process indirectly, primarily by affecting the neural network' s prediction (i.e., the output distribution p_O).

2 Loss Function $L(\mathbf{x}|c)$

Given the prior parameters $\boldsymbol{\theta}_0$ and the accuracy schedule $\beta(t)$, consider a sequence of n sender samples $\mathbf{y}_1, \ldots, \mathbf{y}_n$ drawn at time steps $t_1, \ldots, t_n$ (where $t_i = i/n$). The sender distribution at step i is $p_S(\cdot|\mathbf{x}; \alpha_i)$, where:

$$\alpha_i = \beta(t_i) - \beta(t_{i-1}) \tag{11}$$

The conditional receiver distribution at step i is $p_R(\cdot|\boldsymbol{\theta}_{i-1}; t_{i-1}, c, \alpha_i)$, and the input parameter sequence $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ is recursively computed as:

$$\boldsymbol{\theta}_i = h(\boldsymbol{\theta}_{i-1}, \mathbf{y}_i, \alpha_i) \tag{12}$$

The n-step discrete-time loss $L^n(\mathbf{x}|c)$ is defined as the expected number of nats required to transmit $\mathbf{y}_1, \ldots, \mathbf{y}_n$, and the reconstruction loss $L^r(\mathbf{x}|c)$ is the expected number of nats required to subsequently transmit $\mathbf{x}$. Due to the use of a bits-back coding scheme, transmitting a sample from p_S to a receiver with p_R requires $\mathrm{DKL}(p_S \| p_R)$ nats. Therefore:

$$L^n(\mathbf{x}|c) = \mathbb{E}_{p(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n-1})} \sum_{i=1}^{n} \mathrm{DKL}(p_S(\cdot|\mathbf{x}; \alpha_i) \| p_R(\cdot|\boldsymbol{\theta}_{i-1}; t_{i-1}, c, \alpha_i)) \tag{13}$$

where

$$p(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n) = \prod_{i=1}^{n} p_U(\boldsymbol{\theta}_i|\boldsymbol{\theta}_{i-1}, \mathbf{x}; \alpha_i) \tag{14}$$

Moreover, since the marginal probability of $\boldsymbol{\theta}_n$ is given by $p_F(\cdot|\mathbf{x}, 1)$ (according to (10) and $\sum \alpha_i = \beta(t_n) - \beta(t_0) = \beta(1)$), the reconstruction loss is:

$$L^r(\mathbf{x}|c) = -\mathbb{E}_{p_F(\boldsymbol{\theta}|\mathbf{x}, 1)}[\ln p_O(\mathbf{x}|\boldsymbol{\theta}, c, 1)] \tag{15}$$

The total loss function $L(\mathbf{x}|c)$ is defined as the total number of nats required to transmit the data, i.e., the sum of the n-step loss and the reconstruction loss:

$$L(\mathbf{x}|c) = L^n(\mathbf{x}|c) + L^r(\mathbf{x}|c) \tag{16}$$

2.1 Trick for Computing the Discrete-Time Loss $L^n(\mathbf{x}|c)$

Equation (13) can be rewritten as:

$$L^n(\mathbf{x}|c) = n\mathbb{E}_{i\sim U\{1,n\}} \mathbb{E}_{p_U(\boldsymbol{\theta}_1|\boldsymbol{\theta}_0, \mathbf{x}; \alpha_1)} \cdots \mathbb{E}_{p_U(\boldsymbol{\theta}|\boldsymbol{\theta}_{i-2}, \mathbf{x}; \alpha_{i-1})} \mathrm{DKL}(p_S(\cdot|\mathbf{x}; \alpha_i) \| p_R(\cdot|\boldsymbol{\theta}; t_{i-1}, c, \alpha_i)) \tag{17}$$

where $U\{1, n\}$ denotes the uniform distribution over integers from 1 to n. Moreover, from equations (8) and (10), we obtain:

$$\mathbb{E}_{p_U(\boldsymbol{\theta}_1|\boldsymbol{\theta}_0, \mathbf{x}; \alpha_1)} \cdots \mathbb{E}_{p_U(\boldsymbol{\theta}|\boldsymbol{\theta}_{i-2}, \mathbf{x}; \alpha_{i-1})} = \mathbb{E}_{p_U(\boldsymbol{\theta}|\boldsymbol{\theta}_0, \mathbf{x}; \beta(t_{i-1}))} \tag{18}$$

$$= \mathbb{E}_{p_F(\boldsymbol{\theta}|\mathbf{x}; t_{i-1})} \tag{19}$$

Therefore:

$$L^n(\mathbf{x}|c) = n\mathbb{E}_{i\sim U\{1,n\},\, p_F(\boldsymbol{\theta}|\mathbf{x}; t_{i-1})}[\mathrm{DKL}(p_S(\cdot|\mathbf{x}; \alpha_i) \| p_R(\cdot|\boldsymbol{\theta}; t_{i-1}, c, \alpha_i))] \tag{20}$$

This formulation allows us to approximate $L^n(\mathbf{x}|c)$ via Monte Carlo sampling, without the need to compute an explicit n-step sum.

2.2 Continuous-Time Loss $L^\infty(\mathbf{x}|c)$

Inspired by Variational Diffusion Models (6), we derive the continuous-time loss function $L^\infty(\mathbf{x}|c)$ by taking the limit of $L^n(\mathbf{x}|c)$ as $n \to \infty$. Define:

$$\epsilon \stackrel{\text{def}}{=} \frac{1}{n} \tag{21}$$

$$\alpha(t, \epsilon) \stackrel{\text{def}}{=} \beta(t) - \beta(t - \epsilon) \tag{22}$$

$$L^\infty(\mathbf{x}|c) \stackrel{\text{def}}{=} \lim_{n\to\infty} L^n(\mathbf{x}|c) \tag{23}$$

Then, according to the definition of $L^n(\mathbf{x}|c)$ in equation (20), we obtain:

$$L^\infty(\mathbf{x}|c) = \lim_{\epsilon\to 0} \frac{1}{\epsilon} \mathbb{E}_{t\sim U(\epsilon,1),\, p_F(\boldsymbol{\theta}|\mathbf{x}, t-\epsilon)} \left[\mathrm{DKL}(p_S(\cdot|\mathbf{x}; \alpha(t, \epsilon)) \| p_R(\cdot|\boldsymbol{\theta}; t - \epsilon, c, \alpha(t, \epsilon)))\right] \tag{24}$$

where $U(a, b)$ denotes the continuous uniform distribution over the interval $[a, b]$. For all sender–receiver distribution pairs considered in this work, it can be shown (following a derivation similar to that in the original paper) that:

$$\mathrm{DKL}(p_S(\cdot|\mathbf{x};\alpha)\|p_R(\cdot|\boldsymbol{\theta};t,c,\alpha)) = \sum_{d=1}^{D} \mathrm{DKL}(\mathcal{N}(g(x^{(d)}), C\alpha^{-1})\|P^{(d)}(\boldsymbol{\theta},c,t) * \mathcal{N}(0, C\alpha^{-1})) \tag{25}$$

Here, $g : \mathcal{X} \to \mathcal{Y}$ is a function from data space to sender space, $P^{(d)}(\boldsymbol{\theta}, c, t)$ is a distribution over $\mathcal{Y}$ with finite mean and variance, $*$ denotes convolution between probability distributions, and C is a scalar constant.

Using Proposition 3.1 from the original paper (1) , when $\alpha(t,\epsilon)^{-1} \to \infty$ (i.e., $\epsilon \to 0$), we have:

$$\lim_{\epsilon\to 0} \mathrm{DKL}(p_S(\cdot|\mathbf{x};\alpha(t,\epsilon))\|p_R(\cdot|\boldsymbol{\theta};t-\epsilon,c,\alpha(t,\epsilon))) = \sum_{d=1}^{D} \mathrm{DKL}\left(\mathcal{N}(g(x^{(d)}), \tfrac{C}{\alpha(t,\epsilon)}) \,\middle\|\, \mathcal{N}(\mathbb{E}[P^{(d)}(\boldsymbol{\theta},c,t-\epsilon)], \tfrac{C}{\alpha(t,\epsilon)})\right) \tag{26}$$

$$= \frac{\alpha(t,\epsilon)}{2C} \left\|g(\mathbf{x}) - \mathbb{E}[P(\boldsymbol{\theta},c,t-\epsilon)]\right\|^2 \tag{27}$$

Therefore,

$$L^{\infty}(\mathbf{x}|c) = \mathbb{E}_{t\sim U(0,1),\, p_F(\boldsymbol{\theta}|\mathbf{x},t)} \left[\lim_{\epsilon\to 0} \frac{\alpha(t,\epsilon)}{\epsilon} \cdot \frac{\left\|g(\mathbf{x}) - \mathbb{E}[P(\boldsymbol{\theta},c,t)]\right\|^2}{2C}\right] \tag{28}$$

Since $\lim_{\epsilon\to 0} \frac{\alpha(t,\epsilon)}{\epsilon} = \frac{d\beta(t)}{dt} = \alpha(t)$, we finally obtain:

$$L^{\infty}(\mathbf{x}|c) = \mathbb{E}_{t\sim U(0,1),\, p_F(\boldsymbol{\theta}|\mathbf{x},t)} \left[\alpha(t) \cdot \frac{\left\|g(\mathbf{x}) - \mathbb{E}[P(\boldsymbol{\theta},c,t)]\right\|^2}{2C}\right] \tag{29}$$

2.3 Conditional Sample Generation

Given prior parameters $\boldsymbol{\theta}_0$, condition c, accuracy values $\alpha_1, \ldots, \alpha_n$, and the corresponding time steps $t_i = i/n$, the n-step sampling process recursively generates $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ as follows:

1. Sample $\mathbf{x}'$ from $p_O(\cdot|\boldsymbol{\theta}_{i-1}, c, t_{i-1})$.
2. Sample $\mathbf{y}$ from $p_S(\cdot|\mathbf{x}', \alpha_i)$ (which implies $\mathbf{y} \sim p_R(\cdot|\boldsymbol{\theta}_{i-1}; t_{i-1}, c, \alpha_i)$).
3. Set $\boldsymbol{\theta}_i = h(\boldsymbol{\theta}_{i-1}, \mathbf{y}, \alpha_i)$.

Repeat the above steps n times. Given $\boldsymbol{\theta}_n$, run the network again and sample the final output from $p_O(\cdot|\boldsymbol{\theta}_n, c, 1)$.

The pseudocode is given below (Algorithm 1):

Algorithm 1 Conditional BFN Sample Generation

Require: Prior parameters $\boldsymbol{\theta}_0$, condition c, accuracy schedule $\beta(t)$ (from which α_i can be derived), number of steps n

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
for $i = 1$ to n **do**
  $t_{i-1} \leftarrow (i-1)/n$
  $\alpha_i \leftarrow \beta(i/n) - \beta((i-1)/n)$
  Sample $\mathbf{x}'$ from $p_O(\mathbf{x}'|\boldsymbol{\theta}, c, t_{i-1})$
  Sample $\mathbf{y}$ from $p_S(\mathbf{y}|\mathbf{x}', \alpha_i)$
  $\boldsymbol{\theta} \leftarrow h(\boldsymbol{\theta}, \mathbf{y}, \alpha_i)$
Sample $\mathbf{x}_{\text{final}}$ from $p_O(\mathbf{x}_{\text{final}}|\boldsymbol{\theta}, c, 1)$
return $\mathbf{x}_{\text{final}}$

3 Experiments

To validate the effectiveness of the proposed Conditional Bayesian Flow Networks (Conditional BFNs), we conducted a series of experiments on the dynamically Binarized MNIST dataset. The goal is to assess the model' s generative ability under given class-label conditions and to investigate the evolution of Bayesian flows when guided by conditioning.

3.1 Experimental Setup

We chose dynamically Binarized MNIST as the primary test bed. The dataset contains 60,000 training samples and 10,000 test samples, each image being 28×28 pixels. Unlike static binarization, dynamic binarization performs a Bernoulli sampling from the original grayscale values at every access, introducing richer randomness and data diversity during training. This is implemented in PyTorch via:

```
torch.bernoulli(x.permute(1, 2, 0).contiguous()).int()
```

ensuring the model sees different binarizations of the same digit over the course of training.

Network Architecture. We employ a modified UNet as the neural backbone. Specifically, the UNet used in our version has 128 model channels, each down-sampling level contains two residual blocks, and attention mechanisms (with four heads) are applied at the 8×8 and 16×16 feature resolutions. To accommodate conditional generation, we introduce a label-embedding layer (`torch.nn.Embedding`) that embeds the class labels (0–9) into the same dimensional space as the time embedding; the two embeddings are summed and injected into each UNet module. The probability-parameter input to UNet first passes through a `FourierImageInputAdapter`, which maps the 28×28 single-channel input (representing binary probabilities) into a multichannel feature map suitable for UNet processing. For binary data, we configure the Bayesian flow to handle two classes and set the maximum $\sqrt{\beta}$ to 3. The resulting model contains roughly 25 M learnable parameters.

Training Configuration. We train with the AdamW optimizer using a learning rate of 1×10^{-4}, $\beta_1 = 0.9$, and $\beta_2 = 0.98$. The training batch size is 512, and the validation batch size is 1,000. The model is trained for 10,000 steps with a gradient-clipping max-norm of 5.0. To stabilize training and improve final performance, we apply an exponential moving average (EMA) with a decay rate of 0.9999.

3.2 Conditional Generation Results and Analysis

We first conducted a qualitative evaluation of the sample-generation capability of the Conditional BFN. By providing the trained model with a specified digit label as the conditioning variable c, the model was able to generate the corresponding handwriting numbers. As shown in Figure 2, when the conditions were set to ‘1’ , ‘5’ , and ‘9’ , respectively, the samples generated by the model clearly exhibited the morphological characteristics of the specified digits. For example, digit ‘1’ typically appears concise, digit ‘5’ shows the characteristic curved structure, and digit ‘9’ has a closed upper loop. Crucially, multiple samples generated under the same condition display reasonable morphological variations, indicating that the model not only learns to follow the condition but also captures the inherent diversity of handwritten digits, thereby avoiding mode collapse. All generated samples strictly satisfy the binarization constraint, with pixel values limited to 0 or 1. Compared with the unconditional BFN (Figure 3, showing samples after 10,000 training steps), the Conditional BFN demonstrates markedly better controllability.

From a quantitative standpoint, analysis of the statistical properties of the generated samples reveals differences in pixel density across conditions, consistent with the varying complexity of the digits themselves. For instance, samples conditioned on ‘1’ have an average pixel density of about 5.5%, whereas those conditioned on ‘5’ average around 11.5%; digit ‘9’ lies between these two, at roughly 9.5%. These statistics further corroborate the model' s ability to learn class-specific features.

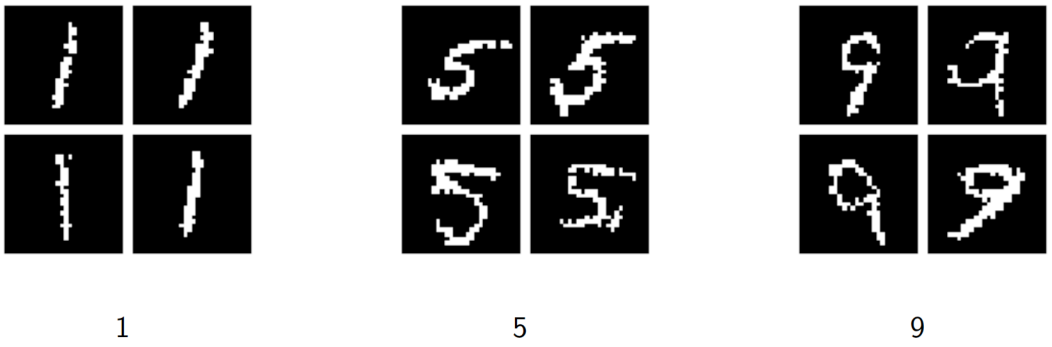


Figure 2: Samples from our modified BFN, illustrating conditional generation for specified digits.

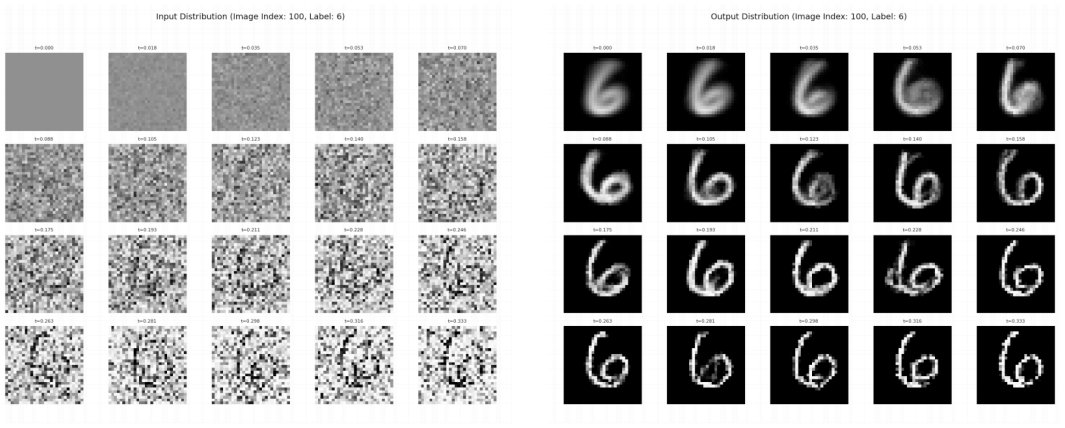
Figure 3: For comparison, unconditional BFN samples generated with the original codebase after 10,000 training steps.

3.3 Visualization of the Bayesian Flow Process

To gain deeper insight into the workings of the Conditional BFN, we visualized the evolution of the Bayesian flow under a fixed label condition. We selected a specific test-set image (index 100, label '6') and observed, at 20 discrete time points from $t = 0$ to $t \approx 1/3$, the evolution of both the input-distribution parameters $p_I(\mathbf{x}|\boldsymbol{\theta})$ and the network output distribution $p_O(\mathbf{x}|\boldsymbol{\theta}, c, t)$. The results are shown in Figure 4.

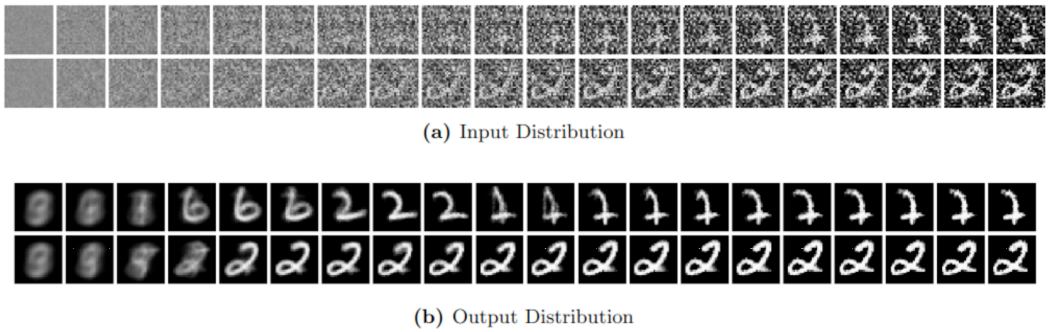
For the input distribution $p_I(\mathbf{x}|\boldsymbol{\theta})$ (Figure 4 (a)), its parameters are near-uniform at the initial moment $t = 0$, reflecting the uninformed prior. As t increases and Bayesian updates incorporate more information, the parameters gradually concentrate from a diffuse state toward a pattern representing the target digit '6'. Around $t \approx 0.1$, a blurred outline of '6' becomes visible and progressively sharpens in subsequent time steps.

By contrast, the evolution of the network output distribution $p_O(\mathbf{x}|\boldsymbol{\theta}, c, t)$ (Figure 4 (b)) exhibits stronger structure and clearer guidance from the condition. Even at early times ($t = 0.018$ and $t = 0.035$), the output distribution is strongly influenced by the label '6', already showing the digit's macroscopic shape, though fine details are not yet crisp. This contrasts with the unconditional setting in the original BFN paper (Figure 5), where early outputs may overlay predictions of multiple digits. As t increases, the input parameters $\boldsymbol{\theta}$ provide more information about the current sample, enabling the network to combine this information with the fixed condition $c =$ '6' to produce increasingly refined and accurate predictions, ultimately yielding a clear digit '6'. This vividly demonstrates the pivotal role of the conditioning variable c in steering the generation process and of the neural network in integrating contextual and conditional information.



(a) Input Distribution (b) Output Distribution

Figure 4: Under the Conditional BFN, evolution of (a) the input distribution and (b) the output distribution over time t for a fixed test image (index 100, label '6') and its corresponding label condition.



(a) Input Distribution

(b) Output Distribution

Figure 5: For comparison, time evolution of input and output distributions in the unconditional setting, reproduced from Figure 13 of the original BFN paper.

3.4 Implementation and Discussion

The forward-propagation logic of the Conditional BFN is similar to that of the original BFN, with the key difference that the conditioning label is incorporated into both the network input and the loss computation. Concretely, after obtaining the input parameters `input_params`, the network `self.net` takes as input the network tensor `net_inputs` (derived from the input parameters), the time step t, and the conditioning label `label` to predict the output parameters `output_params`. The loss function—such as the continuous-time loss `cts_time_loss`—also accounts for these conditioned parameters during computation. During conditional sampling (Algorithm 1), the conditioning label c is used at every step when generating the output distribution p_O.

Preliminary results show that our conditioning mechanism can be effectively integrated into the BFN framework, achieving precise control over the generative process while maintaining good sample diversity and quality. Compared with the unconditional BFN, the Conditional BFN possesses clear advantages in controllability and practical utility. The additional computational overhead introduced by conditioning (mainly from the label embedding and the extra operations in the UNet) is relatively small; in our experiments, total training time increased by less than 3–5%.

Nevertheless, the present work has limitations. Experiments were conducted primarily on the relatively simple MNIST dataset, and the conditioning type was restricted to discrete class labels. Future work could extend to more complex datasets (e.g., experiments on CIFAR-10, CelebA datasets or so on, which we lack the computation resources to do), explore richer conditioning modalities such as text descriptions or image masks, and introduce more comprehensive evaluation metrics (e.g., FID, IS), along with direct performance comparisons against other state-of-the-art conditional generation models.

4 Conclusion

Building upon Bayesian Flow Networks, this paper has proposed and investigated Conditional Bayesian Flow Networks (Conditional BFNs). By introducing a conditioning variable into the neural-network input and correspondingly modifying the output distribution, receiver distribution, and loss functions, we endow the BFN with the ability to guide the generative process via external conditions. We have detailed the mathematical framework of Conditional BFNs, including their discrete-time and continuous-time loss functions, and the conditional sample-generation procedure.

Preliminary experiments on the dynamically binarized MNIST dataset show that the proposed Conditional BFN can effectively generate digit images corresponding to specified class labels and exhibits a reasonable evolution of input and output distributions over time under fixed conditions.

Future work will include more extensive experimental validation on larger and more complex datasets, exploration of diverse conditioning variables, and benchmarking against existing conditional generation models. Beyond that, Conditional BFNs hold promising potential in controllable content generation and cross-modal generation, warranting further investigation.

References

[1] Alex Graves, Rupesh Kumar Srivastava, Timothy Atkinson, and Faustino Gomez. Bayesian Flow Networks. *arXiv preprint arXiv:2308.07037*, 2025. URL: `https://arxiv.org/abs/2308.07037`.

[2] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional Image Generation with PixelCNN Decoders. *arXiv preprint arXiv:1606.05328*, 2016. URL: `https://arxiv.org/abs/1606.05328`.

[3] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. *arXiv preprint arXiv:1807.03039*, 2018. URL: `https://arxiv.org/abs/1807.03039`.

[4] William Harvey, Saeid Naderiparizi, and Frank Wood. Conditional Image Generation by Conditioning Variational Auto-Encoders. *arXiv preprint arXiv:2102.12037*, 2022. URL: `https://arxiv.org/abs/2102.12037`.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. *arXiv preprint arXiv:2006.11239*, 2020. URL: `https://arxiv.org/abs/2006.11239`.

[6] Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational Diffusion Models. *arXiv preprint arXiv:2107.00630*, 2023. URL: `https://arxiv.org/abs/2107.00630`.