# Improving Generalization Capabilities in Conditional Flow Matching

Chelsy Mena González, Solal Peiffer-Smadja, Serhii Vakulenko

Université Jean Monnet / MLDM

January 30, 2026

**Abstract**

Conditional Flow Matching (CFM) has emerged as a robust framework within the broader class of Continuous Normalizing Flows (CNFs), capable of modeling arbitrary probability paths—including those defined by diffusion processes. This paper investigates various regularization strategies to improve the generalization of these paths, specifically evaluating dropout, geometric augmentation, and Carré du Champ (CDC) regularization. Using the Two Moons dataset, we demonstrate that dropout achieves the optimal trade-off between sample quality and generalization, as evidenced by superior Generalization Scores and Memorization Indices. Finally, we extend our analysis to the MNIST dataset, confirming the efficacy of these regularizers in high-dimensional generative tasks.

## 1 Introduction

Flow matching exhibits a tendency to overfit when the target distribution concentrates on low-dimensional manifolds [8]. Rather than learning the continuous manifold structure, the model collapses onto discrete training points, compromising generalization [2]. This phenomenon, often termed *memorization*, represents a fundamental limitation in applying flow-based generative models to scientific datasets where data naturally resides on low-dimensional manifolds embedded in high-dimensional spaces [1].

The first part of the project evaluates these methods on the Two Moons dataset, a fundamental benchmark. We verify that the behavior of the optimal velocity field ($\hat{u}^*$) is fundamentally different in low dimensions (like Two Moons), as explained by [2], compared to high-dimensional settings (like CIFAR-10), as the "collapse" to deterministic trajectories happens late ($t \approx 0.8$), whereas in high dimensions, it happens almost immediately ($t < 0.2$). We also check the "generalization through variance" hypothesis (removing stochasticity from the target actually improves performance and generalization), by comparing model-based regularization (dropout), data-based regularization (geometric augmentation), and geometry-based regularization (CDC) across one nearest neighbor metric.

The second part of the project focuses on the MNIST dataset to observe if the collapse happens faster in higher dimensions and to compare different data generation methods (namely CFM, DDPM and VAE). In order to do this, we tried multiple CFM implementations and ran them

for as long as we could, we also implemented LPIPS as a metric used in order to check whether our digit were close to the original images.

## 2 Regularization in Generative Models

The regularization techniques used can be broadly categorized into three classes:

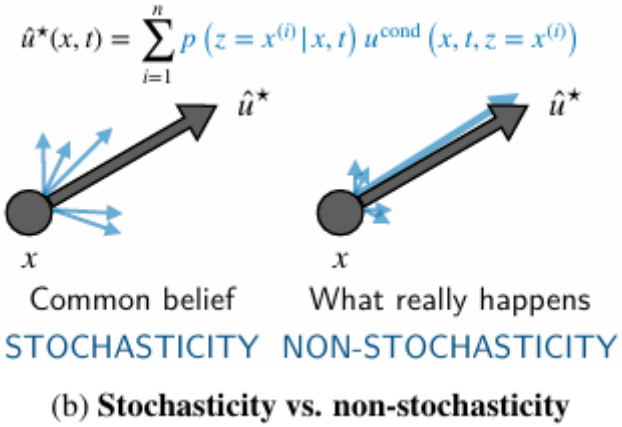
**Model-based regularization** introduces constraints at the architecture level. Dropout [12] prevents feature co-adaptation by randomly deactivating neurons during training.

**Data-based regularization** expands the training distribution through transformations. Geometric augmentation, including rotations and scalings, has been widely used in computer vision [11].

**Geometry-based regularization** incorporates explicit geometric priors. The Carré du Champ method [1] represents a principled approach that injects anisotropic noise aligned with local manifold tangent spaces. This method builds upon diffusion geometry [3] to estimate local covariances that capture the intrinsic data structure.

### 2.1 Manifold Hypothesis and Scientific Applications

The manifold hypothesis states that high-dimensional data often concentrate near lower-dimensional manifolds [4]. This is particularly relevant in scientific applications where physical constraints naturally reduce dimensionality [5]. Figure 1 from [2] illustrates this hypothesis; vector fields converge in the same directions in high dimensions, implying data lies on a lower-dimensional manifold.




(b) **Stochasticity vs. non-stochasticity**

Figure 1: Theoretical interpretation of the empirical results from [2]

# 3 Methodology

## 3.1 Regularization Strategies

We evaluate six configurations on the Two Moons dataset (5,000 samples, 5% Gaussian noise), each addressing memorization through distinct mechanisms:

**Baseline Method:**

- **CFM**: Standard conditional flow matching serves as our baseline.

**Regularization techniques:**

- **Dropout**: We apply stochastic dropout ($p = 0.5$) after each hidden layer in the MLP. This prevents feature co-adaptation and encourages distributed representations.
- **Noise + Augmentation**: We apply random geometric transformations to training samples: rotation ($\theta \sim U[-33°, 33°]$) and scaling ($\alpha \sim U[0.5, 1.5]$). We also apply a Gaussian noise of 0.2.
- **CDC**: Following [1], we compute local covariance matrices $\hat{\Gamma}(x_1)$ from $k = 10$ nearest neighbors. The conditional path becomes:

$$\phi_t^{\text{CDC}}(x_0|x_1) = tx_1 + \left[(1-t)\mathbf{I} + t\hat{\Gamma}(x_1)^{1/2}\right] x_0 \tag{1}$$

  This injects anisotropic noise aligned with the local tangent space, discouraging collapse onto individual points.

## 3.2 Experimental Details

All models (for the Two Moons dataset) use a 3-layer MLP with 64 hidden units and SiLU activations. We train for 1,000 epochs using Adam optimizer with learning rate $10^{-3}$ and batch size 128. For evaluation, we generate 5,000 samples from each trained model and compare against a held-out test set of 1,000 samples.

## 3.3 Evaluation Metrics

We employ three complementary metrics to assess different aspects of performance:

- **Generalization Score (GS)**: $-\log \mathbb{E}_{x_{\text{test}}}[\min_{x_{\text{gen}}} \|x_{\text{test}} - x_{\text{gen}}\|]$, measuring how well generated samples cover the true manifold.
- **Manifold Coverage (MC)**: Percentage of test manifold points within $\epsilon$-radius of generated samples, where $\epsilon$ is set to 0.1 times the manifold diameter.
- **Memorization Index (MI)**: $\frac{1}{N}\sum_i \mathbb{I}(\min_j \|x_{\text{gen}}^{(i)} - x_{\text{train}}^{(j)}\| < \delta)$, with $\delta$ set to 0.05 times the nearest neighbor distance in training data.

The combination of a high GS and a low MI is a sign of overfitting, a combination of high GS and high MI is fitting correctly. MC serves more to confirm MI, as having a low MI should get us a high MC.

# 4 Results

## 4.1 Nearest neighbor metric Dynamics and Training Behavior

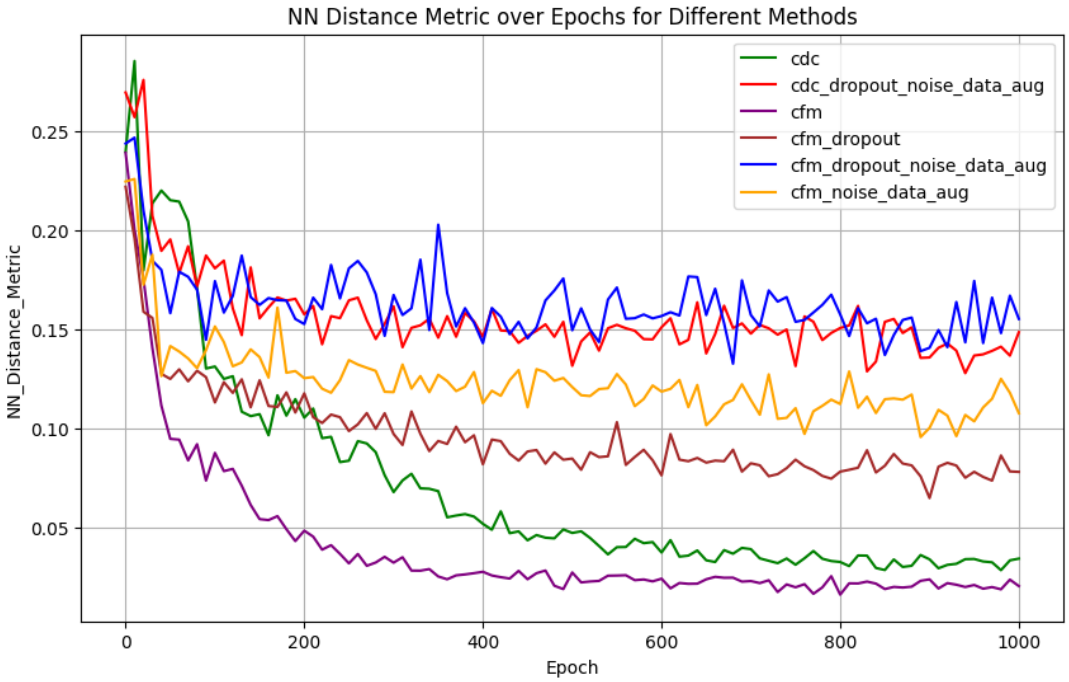



Figure 2: Training average Nearest neighbor metric comparison across methods reveals distinct convergence patterns. Standard CFM and EFM converge rapidly to near-zero, indicating perfect memorization. Regularized methods maintain higher plateaus. The CDC method shows the most stable convergence. The combined approach (All) achieves a balance between stability and smoothing.

Figure 2 illustrates the training average nearest neighbor metric for all methods. First, standard CFM exhibits rapid convergence to near-zero metric values ($< 10^{-3}$) within the first 200 epochs. This indicates successful memorization of the training set. Second, dropout regularization slows convergence but does not prevent eventual overfitting. Third, geometric augmentation achieves a stable metric plateau approximately 10 times higher than the baseline methods, representing the irreducible error introduced by stochastic transformations. Finally, CDC + D + A + N regularization achieves the highest stable plateau.

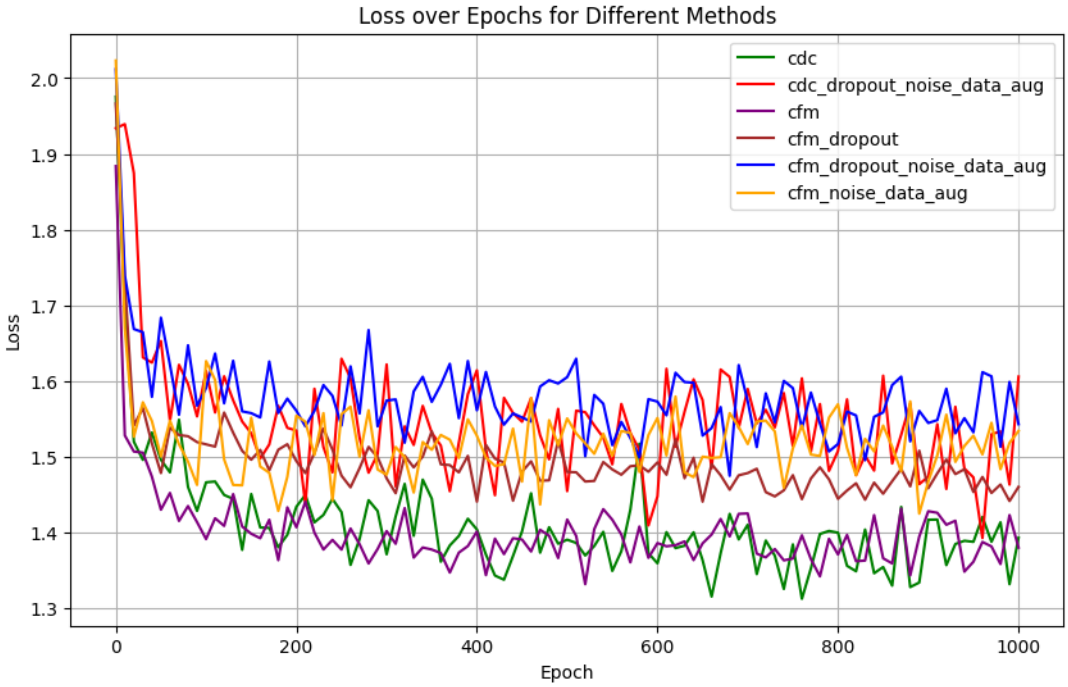



Figure 3: Loss of the different methods

The loss is much more messy and close in terms of value, there isn't much to extract from that plot. The only thing we can say is that we can still see that the same methods are below (namely cdc and cfm) and higher (combined methods CFM + D + N + A and CDC + D + N + A) in values over the epochs.

### 4.2 Visual Results and Quantitative Metrics

| **Method** | **GS** ↑ | **MC (%)** ↑ | **MI** ↓ |
|---|---|---|---|
| CFM | 0.963 | 98.6 | 0.0304 |
| CFM + D | 0.796 | 99.6 | 0.1049 |
| CFM + N + A | 0.740 | 94.0 | 0.1326 |
| CFM + D + N + A | 0.653 | 97.6 | 0.1656 |
| CDC | 0.699 | 93.0 | 0.1472 |
| CDC + D + N + A | 0.663 | 96.4 | 0.1626 |

Table 1: Quantitative comparison at epoch 300.

**Overfitting Analysis (Validity vs. Memorization):** At epoch 300, the baseline **CFM** appears to exhibit signs of overfitting. It achieves a notably high GS (0.963) paired with an extremely low Distance (MI) of 0.0304. While superficially "accurate," this near-zero distance suggests the model has effectively memorized the training set, generating samples that are virtually indistinguishable from the original data points rather than learning a continuous distribution.

**Impact of Regularization:** In contrast, adding regularization forces the model to generalize. The **CFM + D + N + A** configuration yields a lower GS (0.653) and a significantly higher MI (0.1656). This increase in distance is not a failure of the model, but evidence of smoothing:

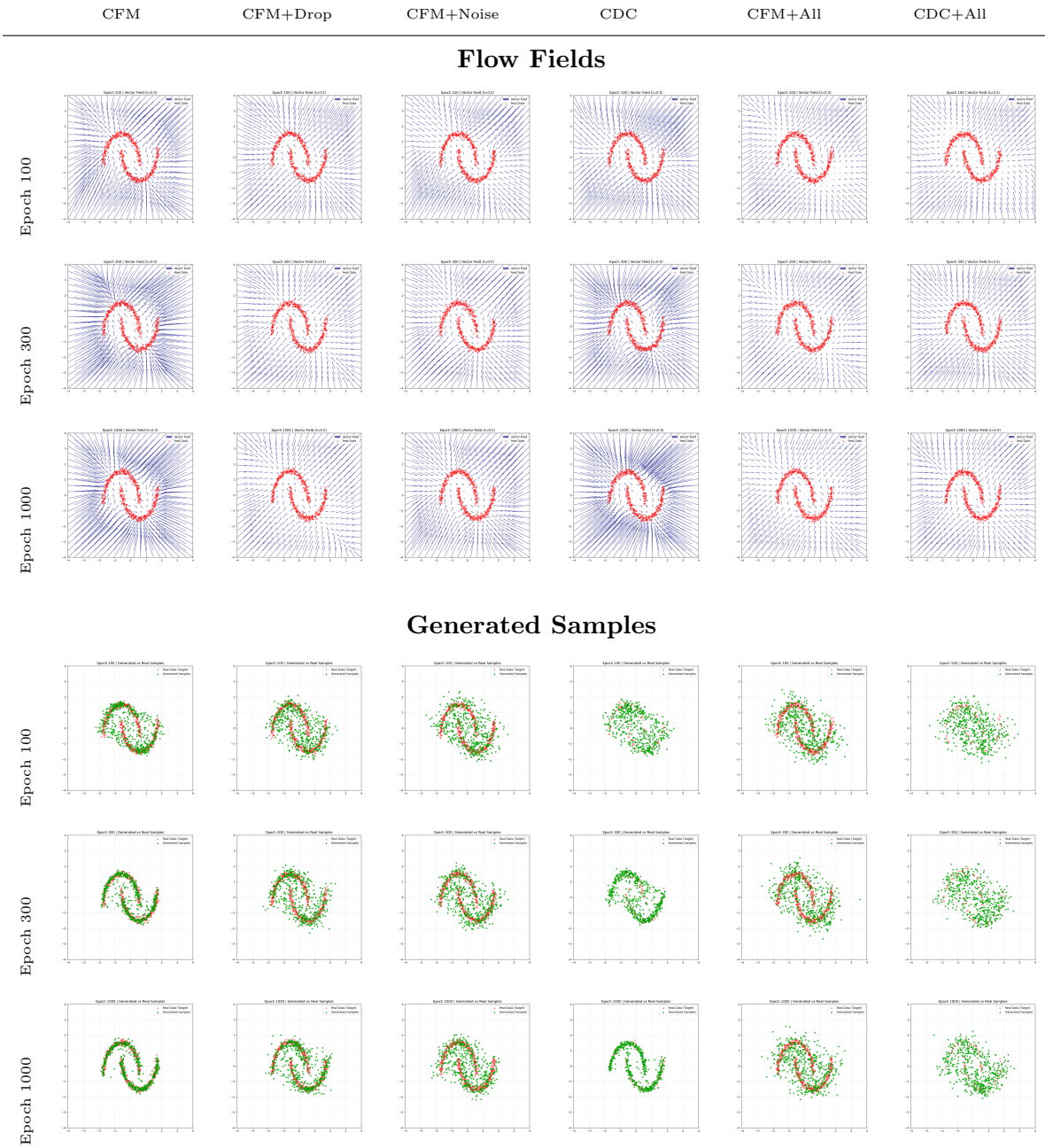



Figure 4: Evolution of Flow and Samples across 100, 300, and 1000 epochs. Note how CDC-based methods maintain better distributional coverage compared to standard CFM variants as training progresses.

the regularized model is generating novel points in the *vicinity* of the manifold rather than collapsing onto specific training examples. This confirms that noise and dropout successfully prevent the "point-wise collapse" seen in the baseline.

**CDC Robustness:** **CDC** is less effective in this context. It has the third lowest GS but the second biggest MI, indicating that the local covariance constraints provide a form of regularization that delays—though may not fully eliminate—the overfitting process compared to the raw baseline.

**Manifold Coverage (Stability):** Despite the regularization trade-offs, coverage remains robust. **CFM + D** achieves the optimal MC (99.6%).

### 4.3 EFM gif

We didn't try EFM for all the metrics since we weren't sure of the implementation. We did do a gif of the result on 500 epoch though, and overfitting is clearly happening before epoch 500 (it needs to be opened with adobe acrobat reader) :

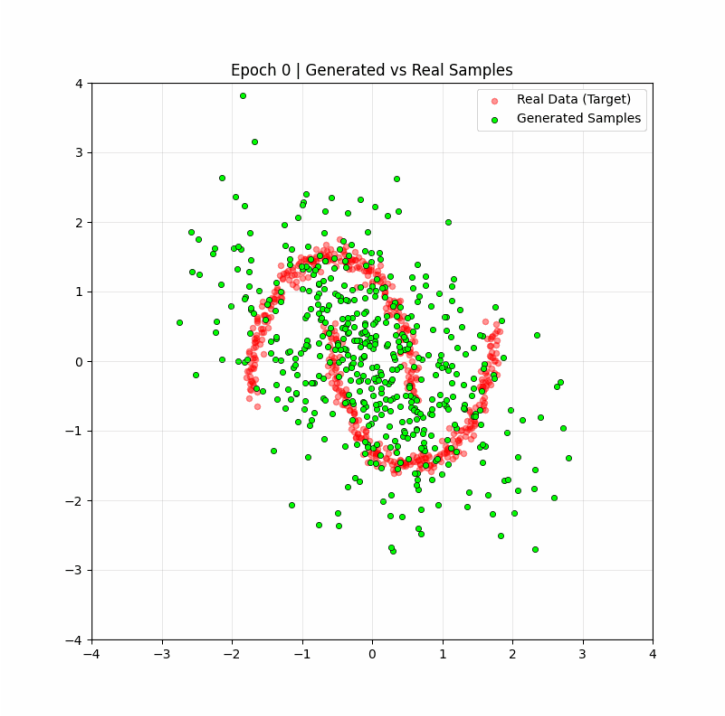



Figure 5: Visualizing the training evolution.

## 5 Two Moons Study Overview

### 5.1 Mechanistic Interpretation

**CDC:** CDC converged slower than CFM; however, it didn't achieve the same kind of plateau as the other methods, which means that it learns the distribution perfectly but simply slower.

**Noise + Augmentation:** Geometric augmentation expands the effective support of the training distribution, preventing the model from fitting to discrete points.

**Dropout:** Dropout prevents feature co-adaptation, encouraging distributed representations. It follows the same kind of plateau as Noise + Augmentation but at a lower value.

**Synergistic Effects:** The Dropout + Noise + Augmentation approach stopped the model from learning on both CFM and CDC, achieving poor scores.

### 5.2 Limitations and Future Directions

While our study provides valuable insights, several limitations warrant consideration. First, the Two Moons dataset represents a simple 2D case. Second, CDC regularization takes more time. Third, systematic hyperparameter tuning was not performed.

## 6 Generalization on MNIST

### 6.1 Dataset

We used the well-known MNIST computer vision dataset [7], consisting of 60,000 grayscale images of handwritten digits (0–9), each 28×28 pixels. It was accessed through the PyTorch package `torchvision.datasets.MNIST`. We performed an 80/20 train/test split resulting in 48000 train images and 12000 validation ones.

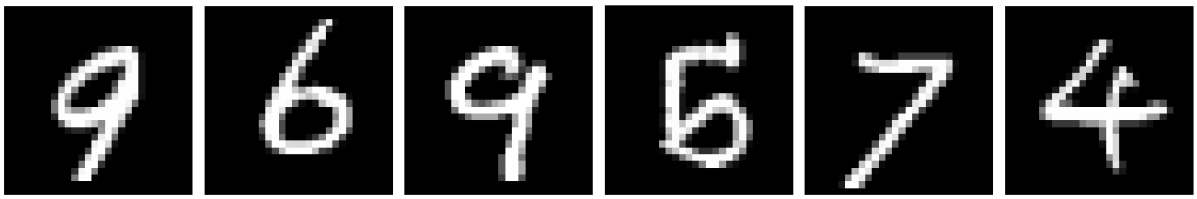


Figure 6: Sample of images from MNIST

### 6.2 Metric

In their work on the generalization ability of the Closed-Form of Flow Matching, Bertrand et al. [2] use the Learned Perceptual Image Patch Similarity (LPIPS) metric to measure the distance of generated samples to the dataset, and measure the creativity of their model as the mean of the LPIPS distances between the generated samples and the datasets. We replicate this approach using the implementation of the metric as made available on GitHub by Rich Zhang [13] with the AlexNet as the feature extractor.

### 6.3 Implementation of Conditional Flow Matching

Our implementation models an arbitrary probability path by learning a time-dependent vector field. Specifically, we utilize the Optimal Transport (OT) formulation proposed by Lipman et al. [9].

#### 6.3.1 Probability Path and Interpolation

The core of our CFM implementation is the definition of the conditional probability path $p_t(x|x_1)$. We implement the Gaussian probability path with a constant-speed vector field. Given a clean data point $x_1 \sim q(x_1)$ and sampled noise $x_0 \sim \mathcal{N}(0, I)$, the interpolated state $x_t$ at time $t \in [0, 1]$ is constructed as:

$$x_t = (1 - t)x_0 + tx_1 + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{2}$$

where $\sigma$ acts as a small regularization parameter to ensure the density does not collapse. In our code, this is realized via the `interpolate` method, which maps the Gaussian distribution to the target manifold via a straight-line trajectory.

### 6.3.2 Vector Field Learning

The objective is to train a neural network $v_\theta(x, t)$ to match the conditional vector field $u_t(x|x_1)$. For the OT path, the target velocity is independent of $t$ and simplifies to the difference between the endpoints:

$$u_t(x|x_1) = x_1 - x_0 \tag{3}$$

The loss function is implemented as the Mean Squared Error (MSE) between the predicted velocity and the target:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,x_1,x_0}[\|v_\theta(x_t, t) - (x_1 - x_0)\|^2] \tag{4}$$

By minimizing this objective, the network $v_\theta$ learns the infinitesimal transformation required to transport mass from the source to the target distribution.

### 6.3.3 Network Architecture

The proposed architecture maintains a constant spatial resolution ($28 \times 28$) throughout the network, avoiding downsampling operations, this is done by the dilation of powers of 3 ($3^0, 3^1, 3^2$). Visualizing this on a 1D strip of pixels:

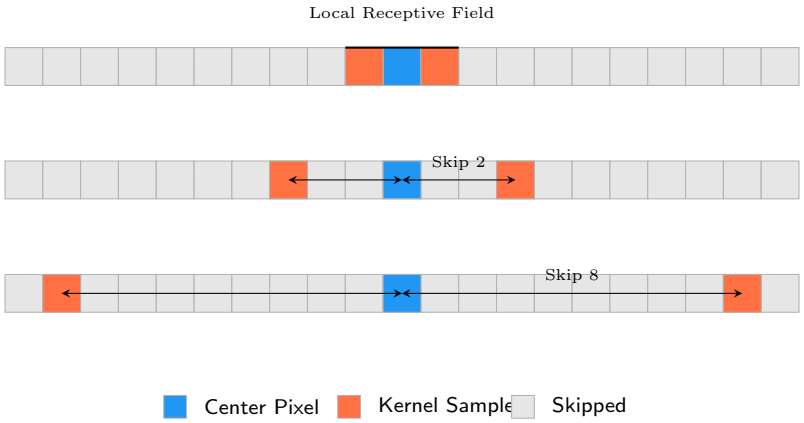



Figure 7: **Visualization of Dilation Rates.** A 1D cross-section showing how the receptive field expands exponentially. While the kernel weights (orange) remain small ($3 \times 3$), the spacing between them increases.

- **Dilation $3^0 = 1$ (Local):** The kernel performs a standard convolution, sampling the current pixel and its immediate neighbors.
  *Skips:* $1 - 1 = 0$ pixels.
- **Dilation $3^1 = 3$ (Mid-Range):** The kernel samples every **3rd** pixel to expand the receptive field.
  *Skips:* $3 - 1 = 2$ pixels between sample points.
- **Dilation $3^2 = 9$ (Global):** The kernel samples every **9th** pixel, reaching across the feature map.
  *Skips:* $9 - 1 = 8$ pixels between sample points.

The workflow below details the global dilation schedule and the internal residual logic.

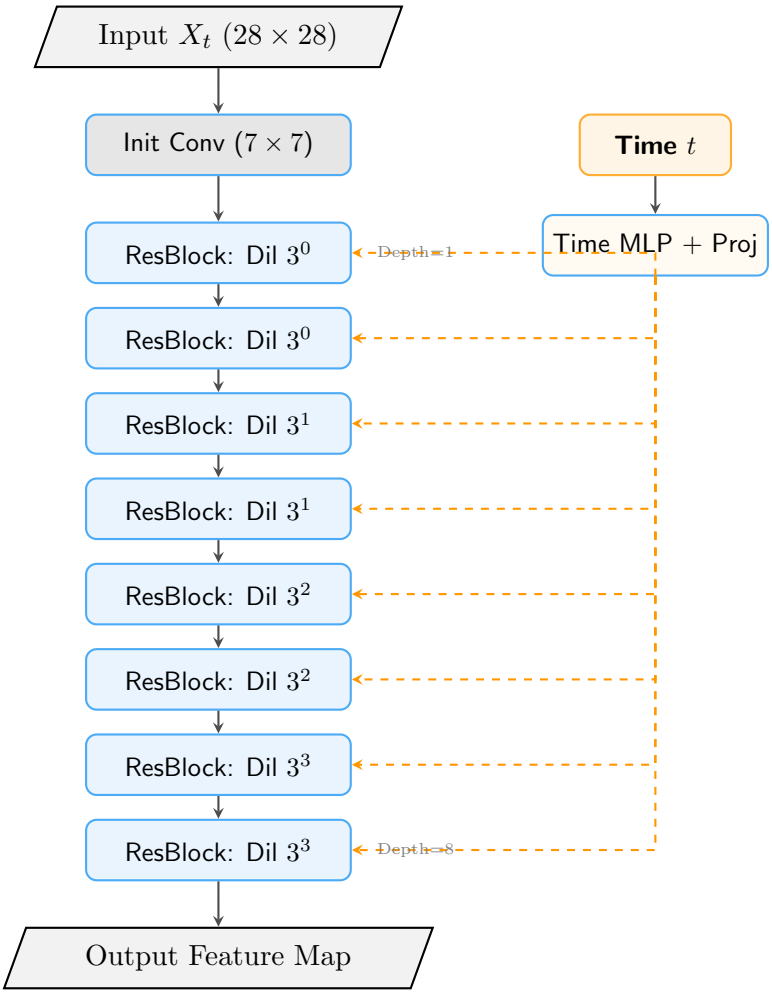



Figure 8: **Macro Workflow.** The 8-layer dilated stack operates on constant spatial dimensions. Time embeddings are projected and injected into every residual block.

The residual block doesn't just process the image; it adds a Time Embedding (vector representing the current diffusion step) to the image features. This tells the layer how much noise it is dealing with right now, here we can see the full residual block logic :

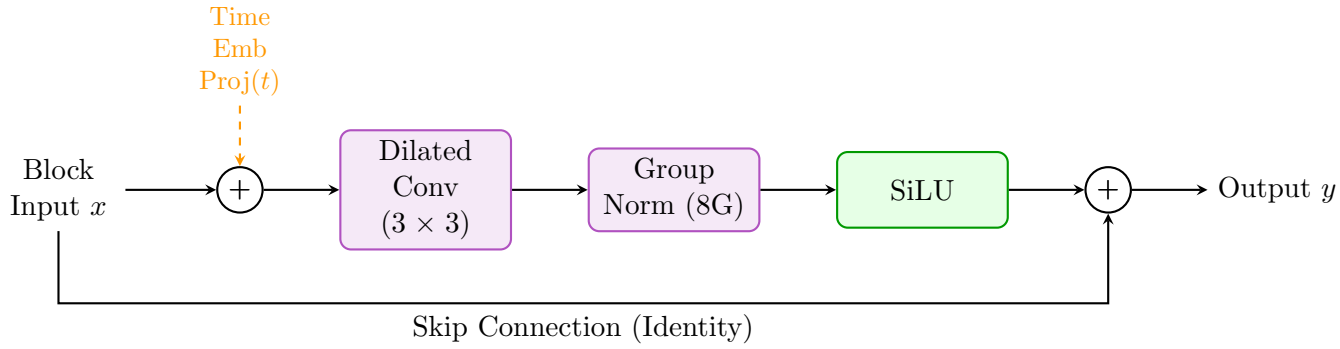



Figure 9: **Residual Block Logic.** Detail of a single layer showing the additive time conditioning and the residual skip connection.

### 6.3.4 Hyperparameters and Training Performance

The specific configuration of the model architecture and the optimization process is summarized in Table 2. The network was optimized using AdamW with a weight decay of $10^{-5}$ to ensure numerical stability and prevent overfitting in the vector field estimation.

Table 2: Hyperparameters for CFM Training on MNIST

| **Hyperparameter** | **Value** |
|---|---|
| Hidden Dimension | 1024 |
| Time Embedding Dimension | 128 |
| Batch Size | 256 |
| Learning Rate | $5 \times 10^{-4}$ |
| Optimizer | AdamW |
| ODE Solver (Inference) | Heun's Method |
| Inference Steps | 50 |
| $\sigma$ (Regularization) | $10^{-4}$ |

Regarding computational performance, the model was trained for 140 epochs. The total training duration was approximately 600 minutes on a single GPU. This runtime includes the periodic evaluation of the Learned Perceptual Image Patch Similarity (LPIPS) metric and the generation of sample grids every five epochs, which didn't account for a significant portion of the time as these were quite fast to run (a few seconds).

### 6.3.5 Results

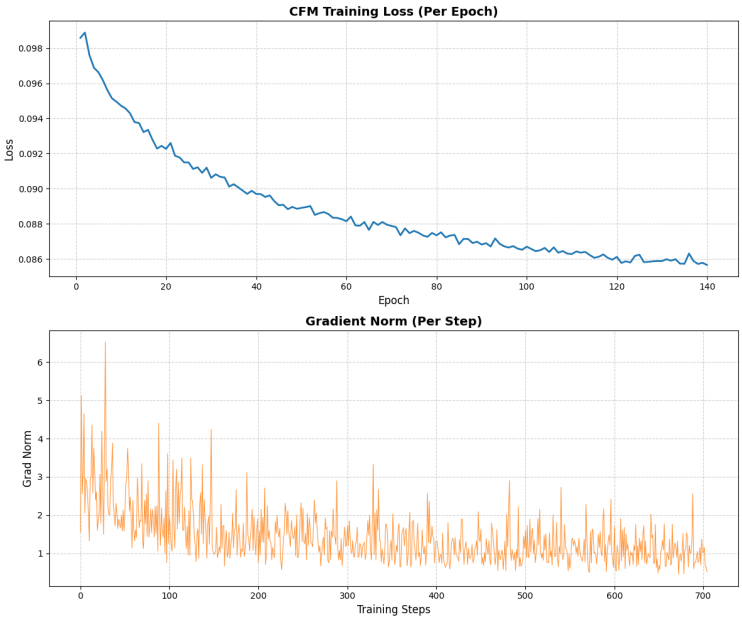



Figure 10: Loss and grad norm per iteration during CFM training

Table 3: LPIPS Distance between 256 samples of the validation set and 256 samples generated with CFM, we can see that the values are pretty low, meaning that maybe we have some overfitting ongoing

| Metric | Value |
|---|---|
| Mean | 0.0821 |
| Std | 0.0242 |
| Min | 0.0392 |
| Max | 0.1747 |

The loss is decreasing with less and less speed, we could have continued the training and maybe reached a plateau of the loss in another 20 epochs or so. However we feel that this should have been enough to see the overfitting happening as it should be faster in higher dimensions compared to lower ones.

## 6.4 Denoising Diffusion Probabilistic Model

Denoising Diffusion Probabilistic models (DDPMs) were introduced by [6], they consist of two stages; the first is a fixed forward diffusion process that gradually adds Gaussian noise to an image, until it is pure noise, and the second is a learned reverse diffusion process where a neural network is trained to gradually denoise an image until you end up with an actual image from the starting point that was pure noise.

Both the forward and reverse process happen for some number of finite time steps $T$, the forward process adds some noise from a Gaussian distribution at each time step $t$, given a sufficiently large $T$ and a schedule for adding noise at each time step, you end up with an isotropic Gaussian distribution at $t = T$ in this gradual process.

### 6.4.1 Network Architecture

To implement it with PyTorch and train it with the dataset, we used the architecture and training parameters as made available in the repository denoising-diffusion-pytorch [10]. Based on the original paper, this package uses a U-Net–based backbone as the denoising network within a Gaussian diffusion framework. The U-Net operates on 28×28 images with 3 input channels and a base feature dimension of 64. Feature widths increase across resolution levels according to multipliers (1, 2, 4), yielding a hierarchical encoder–decoder with skip connections that preserve spatial detail while progressively capturing higher-level representations. This U-Net is embedded in a Denoising Diffusion Probabilistic Model (DDPM) with 1,000 diffusion timesteps for training. During inference, the reverse diffusion process is accelerated by sampling

over 250 timesteps. The model learns to iteratively denoise images corrupted by Gaussian noise, enabling generative sampling from the learned data distribution.

### 6.4.2 Hyperparameters and Training Performance

The specific configuration of the model training and optimization process is summarized in Table 4.

Table 4: Architecture and Training Hyperparameters for DDPM on MNIST

| **Hyperparameter** | **Value** |
|---|---|
| Input Image Size | $28 \times 28$ |
| Input Channels | 3 |
| Diffusion Timesteps (Training) | 1000 |
| Sampling Timesteps | 250 |
| Optimizer | Adam |
| Learning Rate | $1 \times 10^{-4}$ |
| Batch Size | 32 |
| Number of Epochs | 5 |
| Loss Function | DDPM Variational Objective |
| Checkpointing | Every epoch + best model |
| Logging Frequency | Every 100 iterations |

### 6.4.3 Results

We found the model as simple to train as proposed in the original paper. In only an hour we trained 5 epochs and found that the model could generate images of qualitative approximation to the original. In figure 15, we show how in the initial first epoch the model learns most of what it needs to from the dataset and any following epochs don't show significant improvement, the validation loss is quite close to the training loss, which could suggest there is a degree of overfitting in the process. However, as shown in table 7, the mean LPIPS distance between 1000 generated samples and 1000 samples in the validation set is 0.1, a value presented in the work of Bertrand et al. as indicative of good generalization in the CIFAR-10 dataset.

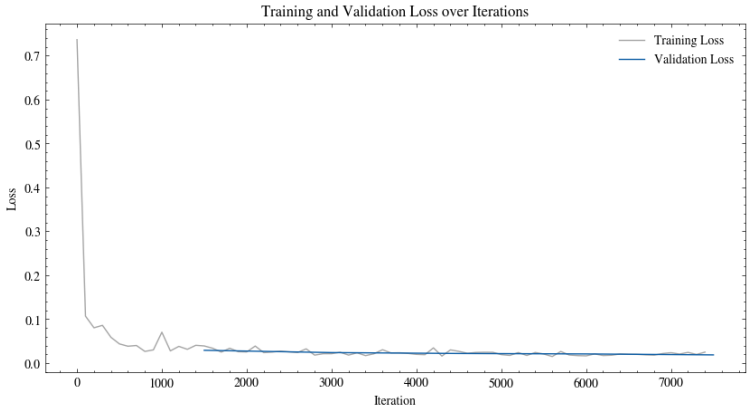



Figure 11: Loss per iteration during DDPM training

Table 5: LPIPS Distance between 1000 samples of the validation set and 1000 samples generated with DDPM

| **Metric** | **Value** |
|---|---|
| Mean | 0.1038 |
| Std | 0.0323 |
| Min | 0.0169 |
| Max | 0.2157 |

Once the best model is selected we can see how the progressive denoising happens as shown in figures 12 and 13, where the mean LPIPS distance between the generated samples and dataset falls the more steps are carried out, and the images become progressively closer to the original dataset. In figure 14 we also show the generated sampled with their closest neighbors from the evaluation dataset to undermark how we are not simply reproducing existing images.

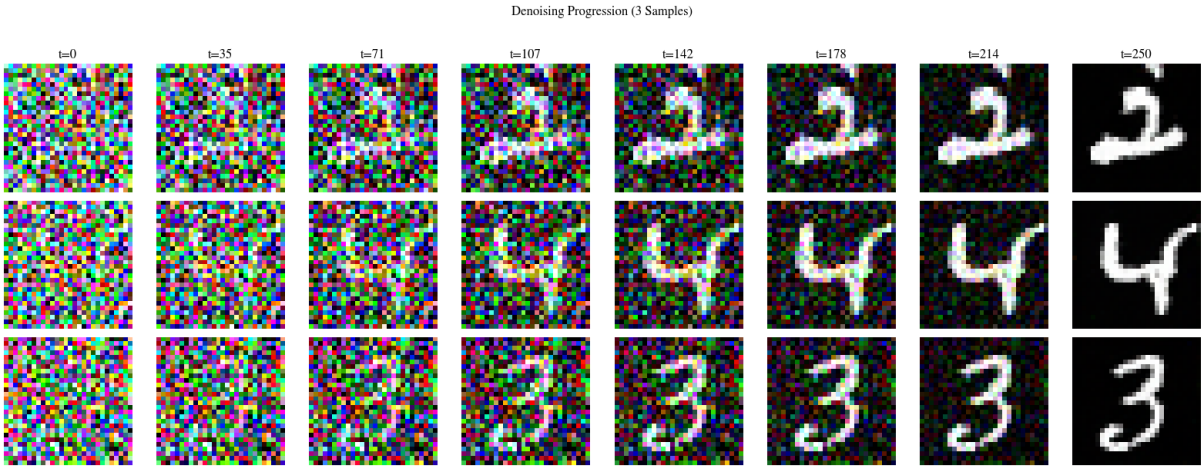



Figure 12: Evolution of Generated Sample Denoising with DDPM

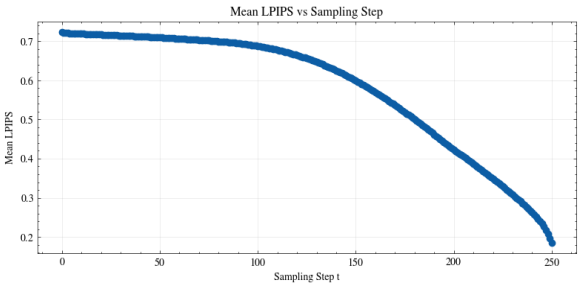



Figure 13: Mean LPIPS distance over DDPM iterations


Figure 14: Closest-neighbor Comparison between Samples generated with DDPM and dataset samples

## 6.5 Implementation of Variational Autoencoder

Our implementation learns a probabilistic latent variable model via the Variational Autoencoder (VAE) framework.

### 6.5.1 Probabilistic Encoding and Reparameterization

The core of our VAE implementation is the construction of the approximate posterior $q_\phi(z|x)$. We implement the encoder as a multi-layer perceptron (MLP) that maps an input $x$ to the parameters of a Gaussian distribution in the latent space. Given an input image $x$, the encoder outputs the mean $\mu$ and log-variance $\log \sigma^2$. To enable backpropagation through the stochastic sampling process, we utilize the reparameterization trick. The latent variable $z$ is sampled as:

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{5}$$

where $\odot$ denotes element-wise multiplication. In our code, this is realized via the `reparameterize` method, which transforms the deterministic encoder outputs and independent noise into a differentiable sample from the approximate posterior.

### 6.5.2 Network Architecture

We employ a Multi-Layer Perceptron (MLP) Variational Autoencoder (VAE) that operates on flattened input data. Our design utilizes fully connected layers, processing the input $28 \times 28$ images as flat 784-dimensional vectors.

The encoder backbone consists of a single fully connected hidden layer with 400 units, followed by a Rectified Linear Unit (ReLU) activation function. To parameterize the approximate posterior distribution, the hidden representation is projected by two parallel fully connected branches into the mean vector ($\mu$) and log-variance vector ($\log \sigma^2$) of the latent space, effectively compressing the input into a 32-dimensional latent representation.

The decoder architecture mirrors the encoder, taking sampled latent vectors $z$ as input. It is composed of a fully connected hidden layer with 400 units and ReLU activation to recover the feature representation. The final layer is a fully connected projection back to the spatial dimension of 784 units ($28 \times 28$ pixels). A Sigmoid activation function is applied to the output to constrain the reconstructed pixel intensities to the range $[0, 1]$.

### 6.5.3 Hyperparameters and Training Performance

The specific configuration of the model architecture and the optimization process is summarized in Table 6. The network was optimized using Adam with a learning rate of $10^{-3}$ to ensure efficient convergence of the generative parameters.

Table 6: Hyperparameters for VAE Training on MNIST

| **Hyperparameter** | **Value** |
|---|---|
| Input Dimension | 784 |
| Hidden Dimension | 400 |
| Latent Dimension | 32 |
| Batch Size | 128 |
| Learning Rate | $1 \times 10^{-3}$ |
| Optimizer | Adam |
| Training Steps | 20,000 |
| Seed | 42 |

Regarding computational performance, the model was trained for 20,000 steps. The total training duration was approximately 12 minutes on a CPU.

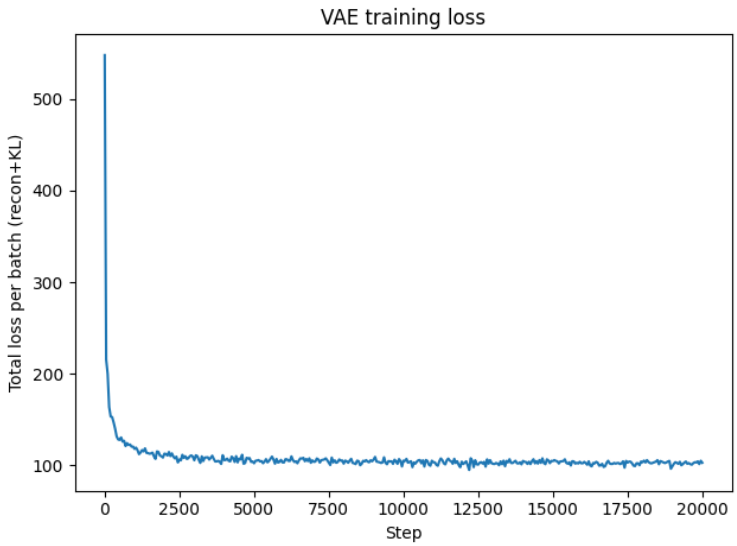



Figure 15: Loss per iteration during VAE training

Table 7: Some Distances between 9 samples of the validation set and 9 samples generated with VAE

| **Metric** | **Value** |
|---|---|
| MSE | 26.6294 |
| SSIM | 0.8413 |
| ORB | 19.62 |
| PSNR | 19.13 dB |

We took 9 samples (one for each digit) from VAE and 9 true images and then calculated these metrics. For ORB we calculated the average distance of matches.

### 6.6 Visual comparison of results


Figure 16: Visual comparison of generated digit samples between methods, on the left is CFM, on the middle is DDPM, on the right is VAE

We can see that some digits aren't really written well in the CFM code, there are some weird creations but overall the digits look the same as the MNIST dataset, in VAE the digits are more blurry (as expected of VAE) and finally DDPM has some weirder lines overall, the digits are really unique, so we could say it overfits less than the CFM.

## 7 Conclusion

Our work can be divided in two main parts, the first part focuses on the Two Moons Dataset, the second part on the MNIST Dataset. The goal of the first part was to show that we could reach overfitting with CFM and try methods to counter that, we implemented three different methods and we did an extensive combination of these methods in order to compare the results. We indeed reached overfitting with the base model and reduced it with the dropout method, however combining all three methods led to no learning at all.

The goal of the second part was to reach overfitting on the MNIST dataset with CFM and then compare results with over kind of data generation models (namely VAE and DDPM). However, we never reached CFM on MNIST (even with different implementations), so we decided to simply compare the different data generation models. Our final plot shows that from the same original dataset, we can get different kind of digits (blurry for VAE, more long accentuated lines for DDPM or noisier for CFM).

## Code

The code to carry out the presented experiments is available in the following repository.

## References

[1] Jacob Bamberger, Iolo Jones, Dennis Duncan, Michael M Bronstein, Pierre Vandergheynst, and Adam Gosztolai. Carré du champ flow matching: better quality-generalisation tradeoff in generative models. *arXiv preprint arXiv:2510.05930*, 2025.

[2] Quentin Bertrand, Anne Gagneux, Mathurin Massias, and Rémi Emonet. On the closed-form of flow matching: Generalization does not arise from target stochasticity. *arXiv preprint arXiv:2506.03719*, 2025.

[3] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006.

[4] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[5] Adam Gosztolai, Giancarlo Loaiza-Ganem, et al. Generative and probabilistic modeling on manifolds for scientific discovery (ms327). Minisymposium at the 17th World Congress on Computational Mechanics (WCCM-ECCOMAS), 2026. Scheduled for July 2026.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.

[7] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]*, 1998.

[8] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2023.

[9] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nicklas, and Mattia Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

[10] Phil Wang (lucidrains). denoising-diffusion-pytorch: Implementation of denoising diffusion probabilistic model in pytorch. https://github.com/lucidrains/denoising-diffusion-pytorch, 2020–. Accessed: 2025-01-29.

[11] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[13] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.