OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport

Derek Onken[*] **Samy Wu Fung**[†] **Xingjian Li**[‡] **Lars Ruthotto**[*‡]

Abstract

A normalizing flow is an invertible mapping between an arbitrary probability distribution and a standard normal distribution; it can be used for density estimation and statistical inference. Computing the flow follows the change of variables formula and thus requires invertibility of the mapping and an efficient way to compute the determinant of its Jacobian. To satisfy these requirements, normalizing flows typically consist of carefully chosen components. Continuous normalizing flows (CNFs) are mappings obtained by solving a neural ordinary differential equation (ODE). The neural ODE's dynamics can be chosen almost arbitrarily while ensuring invertibility. Moreover, the log-determinant of the flow's Jacobian can be obtained by integrating the trace of the dynamics' Jacobian along the flow. Our proposed OT-Flow approach tackles two critical computational challenges that limit a more widespread use of CNFs. First, OT-Flow leverages optimal transport (OT) theory to regularize the CNF and enforce straight trajectories that are easier to integrate. Second, OT-Flow features exact trace computation with time complexity equal to trace estimators used in existing CNFs. On five high-dimensional density estimation and generative modeling tasks, OT-Flow performs competitively to a state-of-the-art CNF while on average requiring one-fourth of the number of weights with 17x speedup in training time and 28x speedup in inference.

1 Introduction

A normalizing flow [44] is an invertible mapping $f\colon \mathbb{R}^d \to \mathbb{R}^d$ between an arbitrary probability distribution and a standard normal distribution whose densities we denote by ρ_0 and ρ_1, respectively. By the change of variables formula, the flow must approximately satisfy [42, 44]

$$\log \rho_0(\boldsymbol{x}) = \log \rho_1(f(\boldsymbol{x})) + \log |\det \nabla f(\boldsymbol{x})| \quad \text{for all} \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{1}$$

Given ρ_0, a normalizing flow is constructed by concatenating invertible layers to form a neural network and training their weights. Since computing the log-determinant in general requires $\mathcal{O}(d^3)$ floating point operations (FLOPS), effective normalizing flows consist of layers whose Jacobians have exploitable structure (e.g., diagonal, triangular, low-rank).

Alternatively, in continuous normalizing flows (CNFs), f is obtained by solving the neural ordinary differential equation (ODE) [11, 23]

$$\partial_t \begin{bmatrix} \boldsymbol{z}(\boldsymbol{x},t) \\ \ell(\boldsymbol{x},t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}\big(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta}\big) \\ \operatorname{tr}\big(\nabla \mathbf{v}(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})\big) \end{bmatrix}, \quad \begin{bmatrix} \boldsymbol{z}(\boldsymbol{x},0) \\ \ell(\boldsymbol{x},0) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix}, \tag{2}$$

for artificial time $t \in [0,T]$ and $\boldsymbol{x} \in \mathbb{R}^d$. The first component maps a point $\boldsymbol{x}$ to $f(\boldsymbol{x}) = \boldsymbol{z}(\boldsymbol{x},T)$ by following the trajectory $\boldsymbol{z}\colon \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$ (Fig. 1). This mapping is invertible and orientation-preserving under mild assumptions on the dynamics $\mathbf{v}\colon \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$. The final state of the

[*]Department of Computer Science, Emory University, donken@emory.edu
[†]Department of Mathematics, University of California, Los Angeles, swufung@math.ucla.edu
[‡]Department of Mathematics, Emory University, {xingjian.li, lruthotto}@emory.edu

Preprint. Under review.

second component satisfies $\ell(\boldsymbol{x}, T) = \log\det \nabla f(\boldsymbol{x})$, which can be derived from the instantaneous change of variables formula as in Chen et al. [11]. Replacing the log determinant with a trace reduces the FLOPS to $\mathcal{O}(d^2)$ for exact computation or $\mathcal{O}(d)$ for an unbiased estimate [19, 23, 57, 58].

To train the dynamics, CNFs minimize the expected negative log-likelihood given by the right-hand-side in (1) [23, 41, 42, 44, 55] via

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{\rho_0(\boldsymbol{x})} \left\{ C(\boldsymbol{x}, T) := \frac{1}{2}\|\boldsymbol{z}(\boldsymbol{x}, T)\|^2 - \ell(\boldsymbol{x}, T) + \frac{d}{2}\log(2\pi) \right\}, \tag{3}$$

where for a given $\boldsymbol{\theta}$, the trajectory $\boldsymbol{z}$ satisfies the neural ODE (2). We note that the optimization problem (3) is equivalent to minimizing the Kullback-Leibler (KL) divergence between ρ_1 and the transformation of ρ_0 given by f (derivation in App. A or [42]).

CNFs are promising but come at considerably high costs. They perform well in density estimation [10, 23, 42] and inference [28, 42], especially in physics and computational chemistry [9, 39]. CNFs are computationally expensive for two predominant reasons. First, even using state-of-the-art ODE solvers, the computation of (2) can require a substantial number of evaluations of $\mathbf{v}$; this occurs, e.g., when the neural network parameters lead to a stiff ODE or dynamics that change quickly in time [3]. Second, computing the trace term in (2) without building the Jacobian matrix is challenging. Using automatic differentiation (AD) to build the Jacobian requires separate vector-Jacobian products for all d standard basis vectors, which amounts to $\mathcal{O}(d^2)$ FLOPS. Trace estimates, used in many CNFs [19, 23, 57, 58], reduce these costs but introduce additional error (Fig. 2). Our approach, OT-Flow, addresses these two challenges.

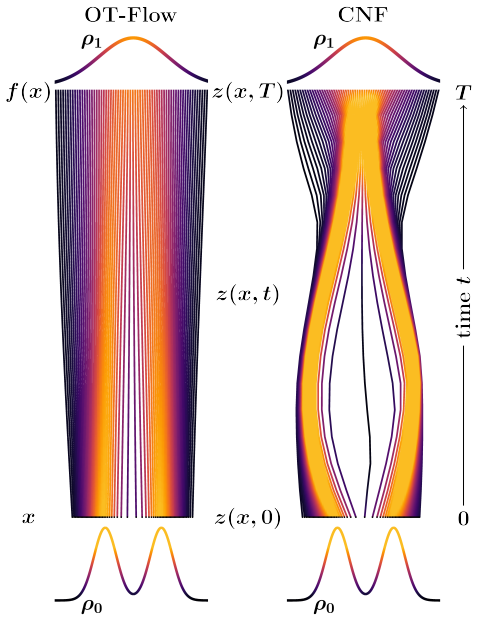


Figure 1: Two flows with approximately equal loss C. OT-Flow enforces straight trajectories. A generic CNF can have more complicated trajectories.

Modeling Contribution Since many flows exactly match two densities while achieving equal loss C (Fig. 1), we can choose a flow that reduces the number of time steps required to solve (2). To this end, we phrase the CNF as an optimal transport (OT) problem by adding transport costs to (3). From this reformulation, we exploit the existence of a potential function whose derivative defines the dynamics $\mathbf{v}$. This potential satisfies the Hamilton-Jacobi-Bellman (HJB) equation, which arises from the optimality conditions of the OT problem. By including an additional cost, which penalizes deviations from the HJB equation, we further reduce the number of necessary time steps to solve (2) (Sec. 2). Ultimately, encoding the underlying regularity of OT into the network absolves it from learning unwanted dynamics, substantially reducing the number of parameters required to train the CNF.

Numerical Contribution To train the flow with reduced time steps, we opt for the discretize-then-optimize approach and use AD for the backpropagation (Sec. 3). Moreover, we analytically derive formulas to efficiently compute the exact trace of the Jacobian in (2). We compute the *exact* Jacobian trace with $O(d)$ FLOPS, matching the time complexity of *estimating* the trace with one Hutchinson vector as used in state-of-the-art CNFs [23]. We demonstrate the competitive runtimes of the trace computation on several high-dimensional examples (Fig. 2). Ultimately, our PyTorch implementation[1] of OT-Flow produces results of similar quality to a state-of-the-art CNF at 17x training and 28x inference speedups on average (Sec. 5).

[1] Open-source code will be available at https://github.com/EmoryMLIP/pyMFGnet .

Table 1: A comparison of flow formulations.

Model	Neural ODEs (2)	Potential Φ	L_2 cost	HJB regularizer
FFJORD [23]	✓	✗	✗	✗
RNODE [19]	✓	✗	✓	✗
Monge-Ampère Flows [58]	✓	✓	✗	✗
Potential Flow Generators [57]	✓	✓	✗	✓
OT-Flow	✓	✓	✓	✓

2 Mathematical Formulation of OT-Flow

Motivated by the similarities between training CNFs and solving OT problems [8, 43], we regularize the minimization problem (3) to encourage straight trajectories (Fig. 1).

Transport Costs We add transport costs $L(\boldsymbol{x}, T)$ to the objective in (3), which results in the regularized problem

$$\min_{\Phi} \ \mathbb{E}_{\rho_0(\boldsymbol{x})} \left\{ C(\boldsymbol{x}, T) + L(\boldsymbol{x}, T) \right\} \quad \text{s.t. (2).} \tag{4}$$

The L_2 transport costs are given by

$$L(\boldsymbol{x}, T) = \int_0^T \frac{1}{2} \|\mathbf{v}(\boldsymbol{z}(\boldsymbol{x}, t), t)\|^2 \, \mathrm{d}t, \tag{5}$$

which penalize the squared arc-length of the trajectories. In practice, this integral can be computed in the ODE solver, similar to the trace accumulation in (2). The OT problem (4) has mathematical properties that we exploit to reduce computational costs [16, 19, 37, 54]. In particular, its solution is unique, and the trajectories matching the two densities ρ_0 and ρ_1 are straight and non-intersecting [20], which reduce the number of time steps to solve (2). The OT formulation also guarantees a solution flow that is smooth, invertible, and orientation preserving [2].

Potential Model Applying the Pontryagin maximum principle [17, 18] to (4) reveals additional structure that guides our modeling. In particular, there exists a potential function $\Phi \colon \mathbb{R}^d \times [0, T] \to \mathbb{R}$ such that

$$\mathbf{v}(\boldsymbol{x}, t; \boldsymbol{\theta}) = -\nabla \Phi(\boldsymbol{x}, t; \boldsymbol{\theta}). \tag{6}$$

Analogous to classical physics, samples move in a manner to minimize their potential.

The optimality conditions of (4) imply that the potential satisfies the HJB equation [16] (App. B), whose violation along the trajectories we penalize by

$$R(\boldsymbol{x}, T) = \int_0^T \left| \partial_t \Phi(\boldsymbol{z}(\boldsymbol{x}, t), t) - \frac{1}{2} \|\nabla \Phi(\boldsymbol{z}(\boldsymbol{x}, t), t)\|^2 \right| \mathrm{d}t. \tag{7}$$

To include this additional regularizer, we parameterize Φ with a neural network instead of $\mathbf{v}$. This HJB regularizer $R(\boldsymbol{x}, T)$ favors plausible Φ without affecting the solution of the optimization problem (4). With implementation similar to $L(\boldsymbol{x}, T)$, the HJB regularizer requires little computation, but drastically simplifies the cost of solving (2) in practice (see examples in App. B and [36, 45, 57]).

OT-Flow Problem In summary, the regularized problem solved in OT-Flow is

$$\min_{\Phi} \ \mathbb{E}_{\rho_0(\boldsymbol{x})} \left\{ C(\boldsymbol{x}, T) + L(\boldsymbol{x}, T) + R(\boldsymbol{x}, T) \right\}, \quad \text{subject to} \quad \text{(2).} \tag{8}$$

Our formulation combines aspects from Grathwohl et al. [23], Zhang et al. [58], Yang and Karniadakis [57], and Finlay et al. [19] (Tab. 1). The computational costs of the regularizers are inexpensive since we accumulate them along the trajectories using the already computed $\nabla \Phi$ (App. C).

3 Implementation of OT-Flow

We define our neural network model, derive analytic formulas for fast and exact trace computations, and describe our efficient ODE solver.

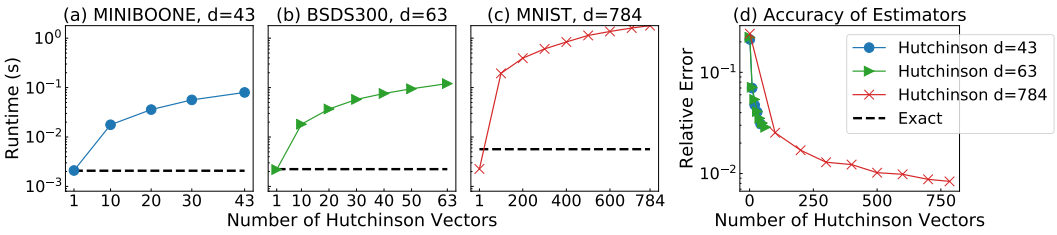


Figure 2: Performance comparison of trace computation using exact approach presented in Sec. 3 and Hutchinson's trace estimator using automatic differentiation. (a-c): runtimes (in seconds) over dimensions 43, 63, and 784, corresponding to the dimensions of MINIBOONE, BSDS300, and MNIST data sets, respectively. (d): relative errors vs. number of Hutchinson vectors for different dimensions.

Network Parameterization We parameterize the potential as

$$\Phi(\boldsymbol{s};\boldsymbol{\theta}) = \boldsymbol{w}^\top N(\boldsymbol{s};\boldsymbol{\theta}_N) + \frac{1}{2}\boldsymbol{s}^\top(\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b}^\top\boldsymbol{s} + c, \quad \text{where} \quad \boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{\theta}_N, \boldsymbol{A}, \boldsymbol{b}, c). \tag{9}$$

Here, $\boldsymbol{s} = (\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$ are the input features corresponding to space-time, $N(\boldsymbol{s};\boldsymbol{\theta}_N)\colon \mathbb{R}^{d+1} \to \mathbb{R}^m$ is a neural network chosen to be a residual neural network (ResNet) [25] in our experiments, and $\boldsymbol{\theta}$ consists of all the trainable weights: $\boldsymbol{w} \in \mathbb{R}^m$, $\boldsymbol{\theta}_N \in \mathbb{R}^p$, $\boldsymbol{A} \in \mathbb{R}^{r\times(d+1)}$, $\boldsymbol{b} \in \mathbb{R}^{d+1}$, $c \in \mathbb{R}$. We set a rank $r = \min(10, d)$ to limit the number of parameters of the symmetric matrix $\boldsymbol{A}^\top\boldsymbol{A}$. Here, $\boldsymbol{A}$, $\boldsymbol{b}$, and c model quadratic potentials, i.e., linear dynamics; N models the nonlinear dynamics.

ResNet Our experiments use a simple two-layer ResNet. When tuning the number of layers as a hyperparameter, we found that wide networks promoted expressibility but deep networks offered no noticeable improvement. For simplicity, we present the two-layer derivation (for the derivation of a ResNet of any depth, see App. D or [45]). The two-layer ResNet uses an opening layer to convert the $\mathbb{R}^{d+1}$ inputs to the $\mathbb{R}^m$ space, then one layer operating on the features in hidden space $\mathbb{R}^m$

$$\begin{aligned} \boldsymbol{u}_0 &= \sigma(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0) \\ N(\boldsymbol{s};\boldsymbol{\theta}_N) = \boldsymbol{u}_1 &= \boldsymbol{u}_0 + h\,\sigma(\boldsymbol{K}_1\boldsymbol{u}_0 + \boldsymbol{b}_1). \end{aligned} \tag{10}$$

We use step-size h=1, dense matrices $\boldsymbol{K}_0 \in \mathbb{R}^{m\times(d+1)}$ and $\boldsymbol{K}_1 \in \mathbb{R}^{m\times m}$, and biases $\boldsymbol{b}_0, \boldsymbol{b}_1 \in \mathbb{R}^m$. We select the element-wise activation function $\sigma(\boldsymbol{x}) = \log(\exp(\boldsymbol{x}) + \exp(-\boldsymbol{x}))$, which is the antiderivative of the hyperbolic tangent, i.e., $\sigma'(\boldsymbol{x}) = \tanh(\boldsymbol{x})$. Therefore, hyperbolic tangent is the activation function of the flow $\nabla\Phi$.

Gradient Computation The gradient of the potential is

$$\nabla_{\boldsymbol{s}}\Phi(\boldsymbol{s};\boldsymbol{\theta}) = \nabla_{\boldsymbol{s}}N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w} + (\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b}, \tag{11}$$

where we simply take the first d components of $\nabla_{\boldsymbol{s}}\Phi$ to obtain the space derivative $\nabla\Phi$. The first term is computed using chain rule (backpropagation)

$$\begin{aligned} \boldsymbol{z}_1 &= \boldsymbol{w} + h\,\boldsymbol{K}_1^\top \operatorname{diag}(\sigma'(\boldsymbol{K}_1\boldsymbol{u}_0 + \boldsymbol{b}_1))\boldsymbol{w}, \\ \nabla_{\boldsymbol{s}}N(\boldsymbol{s};\boldsymbol{\theta}_N)\,\boldsymbol{w} = \boldsymbol{z}_0 &= \boldsymbol{K}_0^\top \operatorname{diag}(\sigma'(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0))\boldsymbol{z}_1. \end{aligned} \tag{12}$$

Here, $\operatorname{diag}(\boldsymbol{q}) \in \mathbb{R}^{m\times m}$ denotes a diagonal matrix with diagonal elements given by $\boldsymbol{q} \in \mathbb{R}^m$. Multiplication by diagonal matrix is implemented as an element-wise product.

Trace Computation We compute the trace of the Hessian of the potential model. We first note that

$$\operatorname{tr}\left(\nabla^2\Phi(\boldsymbol{s};\boldsymbol{\theta})\right) = \operatorname{tr}\left(\boldsymbol{E}^\top\,\nabla_{\boldsymbol{s}}^2(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w})\,\boldsymbol{E}\right) + \operatorname{tr}\left(\boldsymbol{E}^\top(\boldsymbol{A}^\top\boldsymbol{A})\,\boldsymbol{E}\right), \tag{13}$$

where the columns of $\boldsymbol{E} \in \mathbb{R}^{(d+1)\times d}$ are given by the first d standard basis vectors in $\mathbb{R}^{d+1}$. All matrix multiplications with $\boldsymbol{E}$ can be coded as constant-time indexing operations. The trace of the $\boldsymbol{A}^\top\boldsymbol{A}$ term is trivial. We solve the ResNet term via

$$\operatorname{tr}\left(\boldsymbol{E}^\top\,\nabla_{\boldsymbol{s}}^2(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w})\,\boldsymbol{E}\right) = t_0 + h\,t_1, \quad \text{where}$$

$$\begin{aligned} t_0 &= \left(\sigma''(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0) \odot \boldsymbol{z}_1\right)^\top \left((\boldsymbol{K}_0\boldsymbol{E}) \odot (\boldsymbol{K}_0\boldsymbol{E})\right)\boldsymbol{1}, \\ t_1 &= \left(\sigma''(\boldsymbol{K}_1\boldsymbol{u}_0 + \boldsymbol{b}_1) \odot \boldsymbol{w}\right)^\top \left((\boldsymbol{K}_1\nabla_{\boldsymbol{s}}\boldsymbol{u}_0^\top) \odot (\boldsymbol{K}_1\nabla_{\boldsymbol{s}}\boldsymbol{u}_0^\top)\right)\boldsymbol{1}, \end{aligned} \tag{14}$$

where $\odot$ is the element-wise product of equally sized vectors or matrices, $\mathbf{1} \in \mathbb{R}^d$ is a vector of all ones, and $\nabla_{\boldsymbol{s}} \boldsymbol{u}_0^\top = \boldsymbol{K}_0^\top \sigma'(\boldsymbol{K}_0 \, \boldsymbol{s} + \boldsymbol{b}_0)$. For deeper ResNets, the Jacobian term $\nabla_{\boldsymbol{s}} \boldsymbol{u}_{i-1}^\top \in \mathbb{R}^{m \times (d+1)}$ can be updated and over-written at a computational cost of $\mathcal{O}(m^2 \cdot d)$ FLOPS (App. D).

The trace computation of the first layer uses $\mathcal{O}(m \cdot d)$ FLOPS, and each additional layer uses $\mathcal{O}(m^2 \cdot d)$ FLOPS (App. D). Thus, our exact trace computation has $\mathcal{O}(d)$ time complexity. In clocktime, the analytic exact trace computation is competitive with the Hutchinson's estimator using AD, while introducing no estimation error (Fig. 2). Our efficiency in trace computation (14) stems from exploiting the identity structure of matrix E and not building the full Hessian.

ODE Solver For the forward propagation, we use Runge-Kutta 4 with equidistant time steps to solve (2) as well as the time integrals (5) and (7). The number of time steps is a hyperparameter (App. C). For the backpropagation, we use AD. This technique corresponds to the discretize-then-optimize (DTO) approach, an effective method for ODE-constrained optimization problems [1, 6, 12, 33]. In particular, DTO is efficient for solving neural ODEs [22, 34, 40]. Our implementation exploits the benefits of our proposed exact trace computation combined with the efficiency of DTO.

4 Related Works

Finite Flows Normalizing flows [31, 42, 44, 50] use a concatenation of discrete transformations, where specific architectures are chosen to allow for efficient inverse and Jacobian determinant computations. NICE [13], RealNVP [14], IAF [30], and MAF [41] use either autoregressive or coupling flows where the Jacobian is triangular, so the Jacobian determinant can be tractably computed. GLOW [29] expands upon RealNVP by introducing an additional invertible convolution step. These flows are based on either coupling layers or autoregressive transformations, whose tractable invertibility allows for density evaluation and generative sampling. Neural Spline Flows [15] use splines instead of the coupling layers used in GLOW and RealNVP. Using monotonic neural networks, NAF [26] require positivity of the weights. UMNN [55] circumvent this requirement by parameterizing the Jacobian and then integrating numerically.

Infinitesimal Flows Modeling flows with differential equations is a natural and common concept [38, 46, 49, 56]. In particular, CNFs [10, 11, 23] model their flow via the neural ODE in (2).

FFJORD [23], our baseline, is a state-of-the-art CNF. To alleviate the expensive training costs of CNFs, FFJORD sacrifices the exact but slow trace computation in (2) for a Hutchinson's trace estimator with complexity $\mathcal{O}(d)$ [27]. This estimator helps FFJORD achieve training tractability by reducing the trace cost from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$ per time step. However, during inference, FFJORD has $\mathcal{O}(d^2)$ trace computation cost since CNF inference requires the exact trace (Sec. 1). FFJORD also uses the optimize-then-discretize (OTD) method and an adjoint-based backpropagation where the intermediate gradients are recomputed. In contrast, our exact trace computation is competitive with FFJORD's trace approach during training and faster during inference (Fig. 2). OT-Flow uses DTO with AD for the backpropagation. This combination has been shown to converge quicker when training neural ODEs due to accurate gradient computation, storing intermediate gradients, and fewer time steps [22, 34, 40] (Sec 3).

Flows Influenced by Optimal Transport To encourage straight trajectories, RNODE [19] regularizes FFJORD with transport costs $L(\boldsymbol{x}, T)$ and the Frobenius norm of the Jacobian; they report a 2.8x speedup. Other approaches similarly draw from OT theory but parameterize a potential function [57, 58]. Most similar to us, Potential Flow Generators [57] motivate their model from OT and use the HJB regularizer (7). OT-Flow's modeling combines ideas from these regularized formulations [19, 57, 58] (Tab. 1); OT-Flow's numerics differ substantially (Sec. 3). OT has also been used in other generative models [4, 32, 37, 47, 48, 51].

5 Numerical Experiments

We perform density estimation on seven two-dimensional toy problems and five high-dimensional problems from real data sets. We also demonstrate OT-Flow's generative abilities on MNIST.

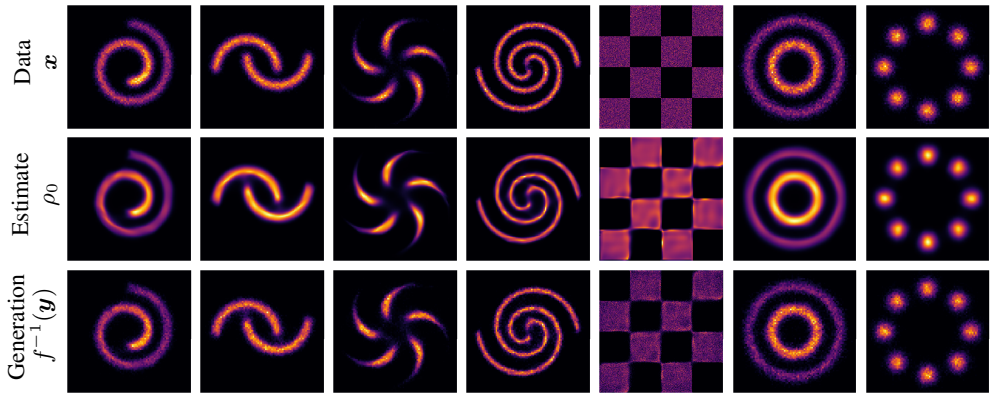


Figure 3: Density estimation on two-dimensional toy problems. **Top:** samples from the unknown distribution. **Middle:** density estimate for unknown ρ_0 computed by inverse flowing from ρ_1 via (2). **Bottom:** samples generated by inverse flow where $\boldsymbol{y} \sim \rho_1(\boldsymbol{y})$.

Metrics In density estimation, the goal is to approximate ρ_0 using observed samples $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i$ are drawn from the distribution ρ_0. In real applications, we lack a ground-truth ρ_0, rendering proper evaluation of the density itself untenable. However, we can follow evaluation techniques applied to generative models. Drawing random points $\{\boldsymbol{y}_i\}_{i=1}^M$ from ρ_1, we invert the flow to generate synthetic samples $\boldsymbol{Q} = \{\boldsymbol{q}_i\}_{i=1}^M$, where $\boldsymbol{q}_i = f^{-1}(\boldsymbol{y}_i)$. We compare the known samples to the generated samples via maximum mean discrepancy (MMD) [24, 35, 43, 52]

$$\mathrm{MMD}(\boldsymbol{X}, \boldsymbol{Q}) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} k(\boldsymbol{q}_i, \boldsymbol{q}_j) - \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} k(\boldsymbol{x}_i, \boldsymbol{q}_j), \quad (15)$$

for Gaussian kernel $k(\boldsymbol{x}_i, \boldsymbol{q}_j) = \exp(-\frac{1}{2}\|\boldsymbol{x}_i - \boldsymbol{q}_j\|^2)$. MMD tests the difference between two distributions (ρ_0 and our estimate of ρ_0) on the basis of samples drawn from each ($\boldsymbol{X}$ and $\boldsymbol{Q}$). A low MMD value means that the two sets of samples are likely to have been drawn from the same distribution [24]. Since MMD is not used in the training, it provides an external, impartial metric to evaluate our model on the hold-out test set (Tab. 2).

Many normalizing flows use C for evaluation. The loss C is used to train the forward flow to match ρ_1. Testing loss, i.e., C evaluated on the testing set, should provide the same quantification on a hold-out set. However, in some cases, the testing loss can be low although the distribution of the flowed samples $f(\boldsymbol{x})$ differs substantially from ρ_1 (Fig. A2 and Fig. A3 in Appendix). Furthermore, because the model's inverse contains error, accurately mapping to ρ_1 with the forward flow does not necessarily mean the inverse flow accurately maps to ρ_0.

Testing loss varies drastically with the integration computation [40, 52, 55]. It depends on ℓ, which is computed along the characteristics via time integration of the trace (App. E). Too few discretization points leads to an inaccurate integration computation and greater inverse error. Thus, a low inverse error implies an accurate integration computation because the flow closely models the ODE. An adaptive ODE solver alleviates this concern when provided a sufficiently small tolerance [23]. Similarly, we check that the flow models the continuous solution of the ODE by computing the inverse error

$$\mathbb{E}_{\rho_0(\boldsymbol{x})}\|f^{-1}\left(f(\boldsymbol{x})\right) - \boldsymbol{x}\| \quad (16)$$

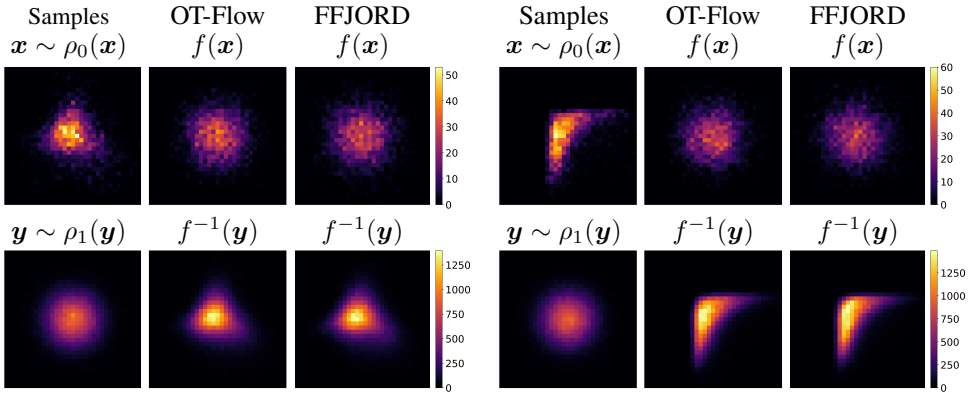
on the testing set using a finer time discretization than used during training. We evaluate the expectation values in (8) and (16) using the discrete samples $\boldsymbol{X}$, which we assume are randomly drawn from and representative of the initial distribution ρ_0.

Toy Problems We train OT-Flow on several two-dimensional toy distributions that serve as standard benchmarks [23, 55, 57]. Given random samples, we train OT-Flow and use the trained model to estimate the density ρ_0 and generate samples (Fig. 3). We also perform a thorough comparison with a state-of-the-art CNF on the toy Gaussian mixture problem (Fig. A2) .

Table 2: Density estimation on real data sets. We trained all models on the same machine (a single NVIDIA TITAN X GPU with 12GB RAM).

Data Set	d	Model	# Param	Training Time (hr)	Testing Time (s)	Inverse Error	MMD
POWER	6	OT-Flow	17K	2.7	8.6	6.93e-8	4.29e-5
		FFJORD	43K	75.6	59.5	1.98e-7	4.37e-5
GAS	8	OT-Flow	69K	3.5	20.6	1.88e-7	5.76e-4
		FFJORD	279K	57.1	171.2	2.71e-7	1.43e-4
HEPMASS	21	OT-Flow	72K	3.6	47.9	2.83e-7	2.01e-5
		FFJORD	547K	51.7	635.6	7.66e-7	2.00e-5
MINIBOONE	43	OT-Flow	78K	0.9	0.9	2.06e-7	2.84e-4
		FFJORD	821K	10.0	27.9	3.59e-7	2.84e-4
BSDS300	63	OT-Flow	297K	8.8	433.3	9.06e-7	9.84e-4
		FFJORD	6.7M	213.2†	34958.8	2.07e-7	1.28e-3

† Training manually terminated before convergence.



(a) MINIBOONE dimension 16 vs 17

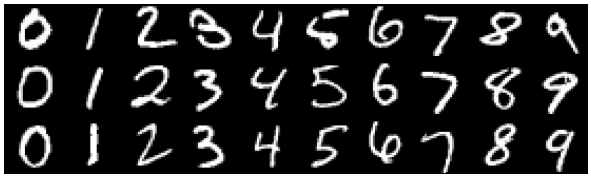
(b) MINIBOONE dimension 28 vs 29

Figure 4: MINIBOONE density estimation. Two-dimensional slices using the 3,648 43-dimensional testing samples $\boldsymbol{x} \sim \rho_0(\boldsymbol{x})$ and 10^5 samples $\boldsymbol{y}$ from distribution ρ_1 (more visuals in Fig. A7).

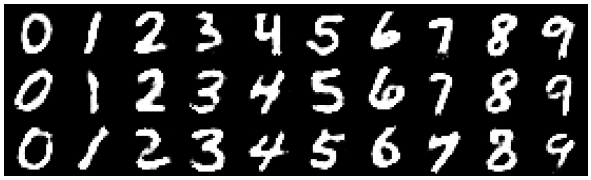
Density Estimation on Real Data Sets We compare our model's performance on real data sets (POWER, GAS, HEPMASS, MINIBOONE) from the University of California Irvine (UCI) machine learning data repository and the BSDS300 data set containing natural image patches. The UCI data sets describe observations from Fermilab neutrino experiments, household power consumption, chemical sensors of ethylene and carbon monoxide gas mixtures, and particle collisions in high energy physics. Prepared by Papamakarios et al. [41], the data sets are commonly used in normalizing flows [14, 23, 26, 29, 55]. The data sets vary in dimensionality (Tab. 2).

For each data set, we compare OT-Flow against the state-of-the-art FFJORD [23] in speed and performance. We compare speed both in training the models and when running the model on the testing set. To compare performance, we compute the MMD between the data set and $M = 10^5$ generated samples $f^{-1}(\boldsymbol{y})$ for each model; for a fair comparison, we use the same $\boldsymbol{y}$ for FFJORD and OT-Flow (Tab. 2). We show visuals of the samples $\boldsymbol{x} \sim \rho_0(\boldsymbol{x})$, $\boldsymbol{y} \sim \rho_1(\boldsymbol{y})$, $f(\boldsymbol{x})$, and $f^{-1}(\boldsymbol{y})$ generated by OT-Flow and FFJORD (Fig. 4, App. F).

The results demonstrate the computational efficiency of OT-Flow relative to FFJORD (Tab. 2). With the exception of the GAS data set, OT-Flow achieves comparable MMD to that of FFJORD with drastically reduced training time. OT-Flow learns a slightly smoothed representation of the GAS data set (Fig. A5). Although we use the exact trace, we introduce little to no extra runtime than computing the Hutchinson's trace estimation with one vector-Jacobian product (Fig. 2). On the testing set, our



(a) Originals



(b) Generatations

Figure 5: MNIST generation conditioned by class. The encoder and decoder are trained beforehand and are responsible for the slight thickness of the generations.

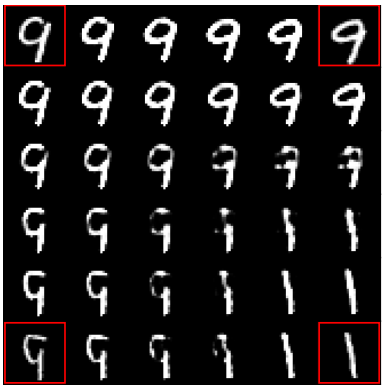


Figure 6: MNIST interpolation in the latent space. Original images are boxed in red.

$\mathcal{O}(d)$ exact trace leads to faster testing time than FFJORD's $\mathcal{O}(d^2)$ approach using AD to compute the exact trace. For example, on the BSDS300 data set, OT-Flow has a testing time of 7 minutes compared to FFJORD's testing time of 9 hours. To evaluate the testing data, we use more time steps than for training, effectively re-discretizing the ODE at different points. The inverse error is near machine precision when adding more time steps, showing that OT-Flow is numerically invertible and suggesting that it approximates the true solution of the ODE. Ultimately, OT-Flow's combination of OT-influenced regularization, reduced parameterization, DTO approach, and efficient exact trace computation results in fast and accurate training and testing.

MNIST We demonstrate the generation quality of OT-Flow using an encoder-decoder structure.

Consider encoder $B\colon \mathbb{R}^{784} \to \mathbb{R}^d$ and decoder $D\colon \mathbb{R}^d \to \mathbb{R}^{784}$ such that $D(B(\boldsymbol{x})) \approx \boldsymbol{x}$. We train d-dimensional flows that map distribution $\rho_0(B(\boldsymbol{x}))$ to ρ_1. The encoder and decoder are each comprised of a single dense layer and activation function (ReLU for B and sigmoid for D). We train the encoder-decoder separate from and prior to training the flows. The trained encoder-decoder, due to its simplicity, renders digits $D(B(\boldsymbol{x}))$ that are a couple pixels thicker than the supplied digit $\boldsymbol{x}$.

We generate new images via two methods. First, using d=64 and a flow conditioned on class, we sample a point $\boldsymbol{y} \sim \rho_1(\boldsymbol{y})$ and map it back to the pixel space to create image $D(f^{-1}(\boldsymbol{y}))$ (Fig. 5b). Second, using d=128 and an unconditioned flow, we interpolate between the latent representations $f(B(\boldsymbol{x}_1)), f(B(\boldsymbol{x}_2))$ of two original images $\boldsymbol{x}_1, \boldsymbol{x}_2$. For interpolated latent vector $\boldsymbol{y} \in \mathbb{R}^d$, we invert the flow and decode back to the pixel space to create image $D(f^{-1}(\boldsymbol{y}))$ (Fig. 6).

6 Discussion

We present OT-Flow, a fast and accurate approach for training and performing inference with CNFs. Our approach tackles two critical computational challenges in CNFs.

First, solving the neural ODEs in CNFs can require many time steps resulting in high computational cost. Leveraging OT theory to regularize the CNF, OT-Flow encourages straight trajectories, leading to ODEs that are easier to solve. In particular, we include transport costs and add an HJB regularizer by exploiting the existence of a potential function. These additions help carry properties from the continuous problem to the discrete problem and allow OT-Flow to use few time steps without sacrificing performance. Second, computing the trace term in (2) is computationally expensive. OT-Flow features exact trace computation at time complexity equal to trace estimators used in existing state-of-the-art CNFs. Our analytic gradient and trace approach is not limited to the ResNet architectures, but expanding to other architectures requires further derivation.

Acknowledgments and Disclosure of Funding

DO and LR are supported by the National Science Foundation award CAREER DMS 1751636, Binational Science Foundation Grant 2018209, and NVIDIA Corporation. SWF is supported by AFOSR MURI FA9550-18-1-0502, AFOSR Grant No. FA9550-18-1-0167, and ONR Grant No. N00014-18-1-2527. Important parts of this research were performed while LR was visiting the Institute for Pure and Applied Mathematics (IPAM), which is supported by the NSF Grant No. DMS-1440415.

References

[1] Feby Abraham, Marek Behr, and Matthias Heinkenschloss. The effect of stabilization in finite element methods for the optimal boundary control of the Oseen equations. *Finite Elements in Analysis and Design*, 41(3):229–251, 2004.

[2] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.

[3] Uri M Ascher. *Numerical methods for evolutionary differential equations*, volume 5. SIAM, 2008.

[4] G. Avraham, Y. Zuo, and T. Drummond. Parallel optimal transport GAN. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4406–4415, 2019.

[5] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):1–34, 2011.

[6] Roland Becker and Boris Vexler. Optimal control of the convection-diffusion equation using stabilized finite element methods. *Numerische Mathematik*, 106(3):349–367, 2007.

[7] J.-D. Benamou, G. Carlier, and F. Santambrogio. Variational mean field games. In *Active particles. Vol. 1. Advances in theory, models, and applications*, Model. Simul. Sci. Eng. Technol., pages 141–171. Birkhäuser/Springer, Cham, 2017.

[8] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.

[9] Johann Brehmer, Felix Kling, Irina Espejo, and Kyle Cranmer. Madminer: Machine learning-based inference for particle physics. *Computing and Software for Big Science*, 4(1):1–25, 2020.

[10] Changyou Chen, Chunyuan Li, Liqun Chen, Wenlin Wang, Yunchen Pu, and Lawrence Carin Duke. Continuous-time flows for efficient inference and density estimation. In *International Conference on Machine Learning (ICML)*, volume 80, pages 824–833, 2018.

[11] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6571–6583, 2018.

[12] S Scott Collis and Matthias Heinkenschloss. Analysis of the streamline upwind/Petrov Galerkin method applied to the solution of optimal control problems. *CAAM TR02-01*, 108, 2002.

[13] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations (ICLR)*, 2015.

[14] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations (ICLR)*, 2017.

[15] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7509–7520, 2019.

[16] Lawrence C Evans. Partial differential equations and Monge-Kantorovich mass transfer. *Current developments in mathematics*, 1997(1):65–126, 1997.

[17] Lawrence C Evans. *Partial Differential Equations*, volume 19. American Mathematical Soc., 2010.

[18] Lawrence C Evans. An introduction to mathematical optimal control theory version 0.2, 2013.

[19] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ODE. *arXiv:2002.02798*, 2020.

[20] Wilfrid Gangbo and Robert J McCann. The geometry of optimal transportation. *Acta Mathematica*, 177(2):113–161, 1996.

[21] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning (ICML)*, pages 881–889, 2015.

[22] Amir Gholaminejad, Kurt Keutzer, and George Biros. ANODE: Unconditionally accurate memory-efficient gradients for neural ODEs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 730–736, 2019.

[23] Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations (ICLR)*, 2019.

[24] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research (JMLR)*, 13(25): 723–773, 2012.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[26] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning (ICML)*, pages 2078–2087, 2018.

[27] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

[28] John Ingraham, Adam Riesselman, Chris Sander, and Debora Marks. Learning protein structure with a differentiable simulator. In *International Conference on Learning Representations (ICLR)*, 2019.

[29] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10215–10224, 2018.

[30] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4743–4751, 2016.

[31] I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[32] Na Lei, Kehua Su, Li Cui, Shing-Tung Yau, and Xianfeng David Gu. A geometric view of optimal transportation and generative model. *Computer Aided Geometric Design*, 68:1–21, 2019.

[33] Günter Leugering, Peter Benner, Sebastian Engell, Andreas Griewank, Helmut Harbrecht, Michael Hinze, Rolf Rannacher, and Stefan Ulbrich. *Trends in PDE constrained optimization*, volume 165. Springer, 2014.

[34] Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research (JMLR)*, 18(1):5998–6026, 2017.

[35] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International Conference on Machine Learning (ICML)*, pages 1718–1727, 2015.

[36] Alex Tong Lin, Samy Wu Fung, Wuchen Li, Levon Nurbekyan, and Stanley J Osher. APAC-Net: Alternating the population and agent control via two neural networks to solve high-dimensional stochastic mean field games. *arXiv:2002.10113*, 2020.

[37] Jingrong Lin, Keegan Lensink, and Eldad Haber. Fluid flow mass transport for generative networks. *arXiv:1910.01694*, 2019.

[38] Radford M Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov chain Monte Carlo*, 2(11):2, 2011.

[39] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365, 2019.

[40] Derek Onken and Lars Ruthotto. Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows. *arXiv:2005.13420*, 2020.

[41] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2338–2347, 2017.

[42] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv:1912.02762*, 2019.

[43] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[44] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, pages 1530–1538, 2015.

[45] Lars Ruthotto, Stanley J. Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences (PNAS)*, 117(17):9183–9193, 2020.

[46] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain Monte Carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning (ICML)*, pages 1218–1226, 2015.

[47] Tim Salimans, Han Zhang, Alec Radford, and Dimitris N. Metaxas. Improving GANs using optimal transport. In *International Conference on Learning Representations (ICLR)*, 2018.

[48] Maziar Sanjabi, Jimmy Ba, Meisam Razaviyayn, and Jason D Lee. On the convergence and robustness of training gans with regularized optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 7091–7101, 2018.

[49] Johan Suykens, Herman Verrelst, and Joos Vandewalle. On-line learning Fokker-Planck machine. *Neural Processing Letters*, 7:81–89, 04 1998.

[50] Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

[51] Akinori Tanaka. Discriminator optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6816–6826, 2019.

[52] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR)*, 2016.

[53] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of tr(f(a)) via stochastic Lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017.

[54] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[55] Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1543–1553, 2019.

[56] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *International Conference on Machine Learning (ICML)*, pages 681–688, 2011.

[57] Liu Yang and George Em Karniadakis. Potential flow generator with L_2 optimal transport regularity for generative models. *arXiv:1908.11462*, 2019.

[58] Linfeng Zhang, Weinan E, and Lei Wang. Monge-Ampère flow for generative modeling. *arXiv:1809.10188*, 2018.

A Derivation of Loss C

Let ρ_0 be the initial density of samples, and $\boldsymbol{z}$ be the trajectories that map samples from ρ_0 to ρ_1. The change in density as we flow a sample $\boldsymbol{x} \sim \rho_0$ at time t is given by the change of variables formula

$$\rho_0(\boldsymbol{x}) = \rho\big(\boldsymbol{z}(\boldsymbol{x}, t)\big) \det\big(\nabla \boldsymbol{z}(\boldsymbol{x}, t)\big), \tag{17}$$

where $\boldsymbol{z}(\boldsymbol{x}, 0) = \boldsymbol{x}$. In normalizing flows, the discrepancy between the flowed distribution at final time T, denoted $\rho(\boldsymbol{x}, T)$, and the normal distribution can be measured using the Kullback-Leibler (KL) divergence

$$\mathbb{D}_{\mathrm{KL}}\big[\, \rho(\boldsymbol{x}, T) \,||\, \rho_1(\boldsymbol{x}) \,\big] = \int_{\mathbb{R}^d} \log\left(\frac{\rho(\boldsymbol{x}, T)}{\rho_1(\boldsymbol{x})}\right) \rho(\boldsymbol{x}, T)\, \mathrm{d}\boldsymbol{x}. \tag{18}$$

Changing variables, and using (17), we can rewrite (18) as

$$\begin{aligned}
&\mathbb{D}_{\mathrm{KL}}\big[\, \rho(\boldsymbol{z}(\boldsymbol{x}, T)) \,||\, \rho_1(\boldsymbol{z}(\boldsymbol{x}, T)) \,\big] \\
&= \int_{\mathbb{R}^d} \log\left(\frac{\rho(\boldsymbol{z}(\boldsymbol{x}, T))}{\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))}\right) \rho(\boldsymbol{z}(\boldsymbol{x}, T)) \det\big(\nabla \boldsymbol{z}(\boldsymbol{x}, T)\big)\, \mathrm{d}\boldsymbol{x}, \\
&= \int_{\mathbb{R}^d} \log\left(\frac{\rho_0(\boldsymbol{x})}{\rho_1\big(\boldsymbol{z}(\boldsymbol{x}, T)\big) \det\big(\nabla \boldsymbol{z}(\boldsymbol{x}, T)\big)}\right) \rho_0(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}, \\
&= \int_{\mathbb{R}^d} \Big[\log\big(\rho_0(\boldsymbol{x})\big) - \log\big(\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))\big) - \log\det\big(\nabla \boldsymbol{z}(\boldsymbol{x}, T)\big)\Big] \rho_0(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}.
\end{aligned} \tag{19}$$

For normalizing flows, we assume ρ_1 is the standard normal

$$\rho_1(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^d}} \exp\left(\frac{-\|\boldsymbol{x}\|^2}{2}\right), \tag{20}$$

which will reduce the term

$$\log\big(\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))\big) = -\frac{1}{2}\|\boldsymbol{z}(\boldsymbol{x}, T)\|^2 - \frac{d}{2}\log(2\pi). \tag{21}$$

Substituting (21) into (19), we obtain

$$\begin{aligned}
\mathbb{D}_{\mathrm{KL}} &= \int_{\mathbb{R}^d} \Big[\log\big(\rho_0(\boldsymbol{x})\big) - \log\det\big(\nabla \boldsymbol{z}(\boldsymbol{x}, T)\big) + \frac{1}{2}\|\boldsymbol{z}(\boldsymbol{x}, T)\|^2 + \frac{d}{2}\log(2\pi)\Big] \rho_0(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \\
&= \int_{\mathbb{R}^d} \Big[\log\big(\rho_0(\boldsymbol{x})\big) + C(\boldsymbol{x}, T)\Big] \rho_0(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \\
&= \mathbb{E}_{\rho_0(\boldsymbol{x})}\left\{\log\big(\rho_0(\boldsymbol{x})\big) + C(\boldsymbol{x}, T)\right\},
\end{aligned} \tag{22}$$

where $C(\boldsymbol{x}, T)$ is defined in (3). Density $\rho_0(\boldsymbol{x})$ is unknown in normalizing flows. Thus, the term $\log(\rho_0(\boldsymbol{x}))$ is dropped, and normalizing flows minimize C alone. Subtracting this constant does not affect the minimizer.

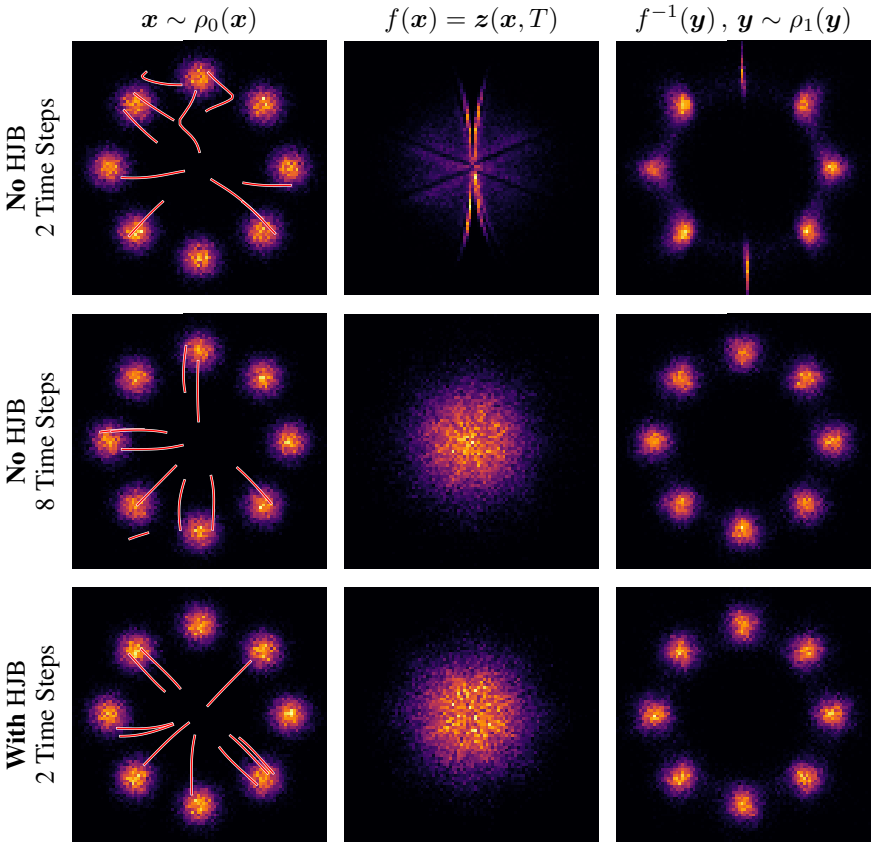


Figure A1: Effect of adding an HJB regularizer during training. The first row presents a flow trained using two RK4 time steps without an HJB regularizer. The second row presents a flow trained using eight RK4 time steps without an HJB regularizer. The third row presents a flow trained using two RK4 time steps *with* an HJB regularizer. For each flow, we show initial, forward mapping, and generation. The HJB regularizer allows for training a flow with one-fourth the number of time steps, leading to a drastic reduction in computational and memory costs. White trajectories display the forward flow f for several random samples; red trajectories display the inverse flow f^{-1}.

B The HJB Regularizer

Theory The optimality conditions of (4) imply that the potential Φ satisfies the Hamilton-Jacobi-Bellman (HJB) [16] equation

$$-\partial_t \Phi(\boldsymbol{x}, t) + \frac{1}{2}\|\nabla\Phi(\boldsymbol{z}(\boldsymbol{x}, t), t)\|^2 = 0, \quad \Phi(\boldsymbol{x}, T) = G(\boldsymbol{x}). \tag{23}$$

where $-\partial_t$ indicates that we solve the equation backwards in time and G is the terminal condition of the partial differential equation (PDE). Consider the KL divergence in (19) after the change of variables is performed. OT theory [7, 54] states that the HJB terminal condition is given by

$$\begin{aligned} G(\boldsymbol{z}(\boldsymbol{x}, T)) &\coloneqq \frac{\delta}{\delta\rho_0}\mathbb{D}_{\mathrm{KL}}\big[\,\rho(\boldsymbol{z}(\boldsymbol{x}, T))\,||\,\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))\,\big] \\ &= \frac{\delta}{\delta\rho_0}\int_{\mathbb{R}^d}\Big[\log\big(\rho_0(\boldsymbol{x})\big) - \log\big(\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))\big) - \log\det\big(\nabla\boldsymbol{z}(\boldsymbol{x}, T)\big)\Big]\rho_0(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\ &= 1 + \log\big(\rho_0(\boldsymbol{x})\big) - \log\big(\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))\big) - \log\det\big(\nabla\boldsymbol{z}(\boldsymbol{x}, T)\big), \end{aligned} \tag{24}$$

where $\frac{\delta}{\delta\rho_0}$ is the variational derivative with respect to ρ_0.

While solving (23) in high-dimensional spaces is notoriously difficult, penalizing its violations along the trajectories is inexpensive. Therefore, we include the value $R(x, T)$ in the objective function,

which we accumulate during the ODE solve (Sec. 2). The density ρ_0, which is required to evaluate G, is unknown in our problems. Similar to Yang and Karniadakis [57], we do not enforce the HJB terminal condition but do enforce the HJB equation for $t \in (0, T)$ via regularizer R.

Effect of Added Regularizer In Fig. A1, we show the effect of training the toy Gaussian mixture problem with and without the HJB regularizer R. For this demonstration, we train the model using two Runge-Kutta 4 (RK4) steps. As a result, the L_2 cost is penalized at too few time steps. Therefore, without an HJB regularizer, the model achieves poor performance and unstraight characteristics (Fig. A1). This issue can be remedied by adding more RK4 time steps or the HJB regularizer. The additional RK4 time steps would add significant memory and computational overhead. The HJB regularizer, however, adds little memory and computation. We thus can train the model with two RK4 time steps and an HJB regularizer with efficient computational cost *and* good performance.

For the demonstration (Fig. A1), we compare three models: two RK4 time steps with no HJB regularizer, eight RK4 time steps with no HJB regularizer, and two RK4 time steps with the HJB regularizer. For several starting points, we plot the forward flow trajectories f in white and the inverse flow f^{-1} trajectories in red. The last two models have straight trajectories, which the first model lacks. All three models are invertible since their forward and inverse trajectories align.

C Implementation Details

We incorporate the accumulation of the regularizers in the ODE. The full optimization problem is

$$\min_{\boldsymbol{\theta}} \;\; \mathbb{E}_{\rho_0(\boldsymbol{x})} \left\{ \alpha_1 C(\boldsymbol{x}, T) + L(\boldsymbol{x}, T) + \alpha_2 R(\boldsymbol{x}, T) \right\} \tag{25}$$

subject to

$$\partial_t \begin{pmatrix} \boldsymbol{z}(\boldsymbol{x},t) \\ \ell(\boldsymbol{x},t) \\ L(\boldsymbol{x},t) \\ R(\boldsymbol{x},t) \end{pmatrix} = \begin{pmatrix} -\nabla\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta}) \\ -\operatorname{tr}(\nabla^2\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})) \\ \frac{1}{2}\|\nabla\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})\|^2 \\ \left|\partial_t\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta}) - \frac{1}{2}\|\nabla\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})\|^2\right| \end{pmatrix}, \quad \begin{pmatrix} \boldsymbol{z}(\boldsymbol{x},0) \\ \ell(\boldsymbol{x},0) \\ L(\boldsymbol{x},0) \\ R(\boldsymbol{x},0) \end{pmatrix} = \begin{pmatrix} \boldsymbol{x} \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

where we optimize the weights $\boldsymbol{\theta}$, defined in (9), that parameterize Φ. We include two hyperparameters α_1, α_2 to assist the optimization. Specially selected hyperparameters can improve the convergence and performance of the model. Other hyperparameters include the hidden space size m, the number of time steps used by the Runge-Kutta 4 solver n_t, the number of ResNet layers for which we use 2 for all experiments, and various settings for the ADAM optimizer.

For validation and testing, we use more time steps than for training, which allows for higher precision and a check that our discrete OT-Flow still approximates the continuous object. A large number of training time steps results in good generalizability and lower inverse error; too few time steps results in high inverse error but low computational cost. We tune the number of training time steps so that we maintain good generalizability (validation and training loss are similar) with low computational cost.

D Exact Trace computation

We expand on the trace computation formulae presented in Sec. 3 for a ResNet with $M+1$ layers.

Gradient Computation To compute the gradient, first note that for an $(M+1)$-layer residual network and given inputs $\boldsymbol{s} = (\boldsymbol{x}, t)$, we obtain $N(\boldsymbol{s}; \boldsymbol{\theta}_N) = \boldsymbol{u}_M$ by forward propagation

$$\begin{aligned} \boldsymbol{u}_0 &= \sigma(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0) \\ \boldsymbol{u}_1 &= \boldsymbol{u}_0 + h\,\sigma(\boldsymbol{K}_1\boldsymbol{u}_0 + \boldsymbol{b}_1) \\ &\vdots \qquad\qquad \vdots \\ \boldsymbol{u}_M &= \boldsymbol{u}_{M-1} + h\,\sigma(\boldsymbol{K}_M\boldsymbol{u}_{M-1} + \boldsymbol{b}_M), \end{aligned} \tag{26}$$

where $h > 0$ is a fixed step size, and the network's weights are $\boldsymbol{K}_0 \in \mathbb{R}^{m\times(d+1)}$, $\boldsymbol{K}_1, \ldots, \boldsymbol{K}_M \in \mathbb{R}^{m\times m}$, and $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_M \in \mathbb{R}^m$.

The gradient of the neural network is computed using backpropagation as follows

$$\begin{aligned}
\boldsymbol{z}_{M+1} &= \boldsymbol{w} \\
\boldsymbol{z}_{M} &= \boldsymbol{z}_{M+1} + h\,\boldsymbol{K}_M^\top \operatorname{diag}\big(\sigma'(\boldsymbol{K}_M\boldsymbol{u}_{M-1}+\boldsymbol{b}_M)\big)\boldsymbol{z}_{M+1}, \\
&\vdots \qquad\qquad \vdots \\
\boldsymbol{z}_{1} &= \boldsymbol{z}_{2} + h\,\boldsymbol{K}_1^\top \operatorname{diag}\big(\sigma'(\boldsymbol{K}_1\boldsymbol{u}_{0}+\boldsymbol{b}_1)\big)\boldsymbol{z}_{2}, \\
\boldsymbol{z}_{0} &= \boldsymbol{K}_0^\top \operatorname{diag}\big(\sigma'(\boldsymbol{K}_0\boldsymbol{s}+\boldsymbol{b}_0)\big)\boldsymbol{z}_{1},
\end{aligned} \tag{27}$$

which gives $\nabla_{\boldsymbol{s}} N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w} = \boldsymbol{z}_0$.

Exact Trace Computation Using (13) and the same $\boldsymbol{E}$, we compute the trace in one forward pass through the layers. The trace of the first ResNet layer is

$$\begin{aligned}
t_0 &= \operatorname{tr}\Big(\boldsymbol{E}^\top \nabla_{\boldsymbol{s}}\big(\boldsymbol{K}_0^\top \operatorname{diag}(\sigma''(\boldsymbol{K}_0\boldsymbol{s}+\boldsymbol{b}_0))\boldsymbol{z}_1\big)\,\boldsymbol{E}\Big) \\
&= \operatorname{tr}\Big(\boldsymbol{E}^\top \boldsymbol{K}_0^\top \operatorname{diag}\big(\sigma''(\boldsymbol{K}_0\boldsymbol{s}+\boldsymbol{b}_0)\odot \boldsymbol{z}_1\big)\,\boldsymbol{K}_0\boldsymbol{E}\Big) \\
&= \big(\sigma''(\boldsymbol{K}_0\boldsymbol{s}+\boldsymbol{b}_0)\odot \boldsymbol{z}_1\big)^\top\big((\boldsymbol{K}_0\boldsymbol{E})\odot(\boldsymbol{K}_0\boldsymbol{E})\big)\mathbf{1},
\end{aligned} \tag{28}$$

using the same notation as (14). For the last step, we used the diagonality of the middle matrix. Computing t_0 requires $\mathcal{O}(m\cdot d)$ FLOPS when first squaring the elements in the first d columns of $\boldsymbol{K}_0$, then summing those columns, and finally one inner product.

To compute the trace of the entire ResNet, we continue with the remaining rows in (27) in reverse order to obtain

$$\operatorname{tr}\Big(\boldsymbol{E}^\top \nabla_{\boldsymbol{s}}^2(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w})\,\boldsymbol{E}\Big) = t_0 + h\sum_{i=1}^{M} t_i, \tag{29}$$

where t_i is computed as

$$\begin{aligned}
t_i &= \operatorname{tr}\Big(J_{i-1}^\top \nabla_{\boldsymbol{s}}\big(\boldsymbol{K}_i^\top \operatorname{diag}(\sigma''(\boldsymbol{K}_i\boldsymbol{u}_{i-1}(\boldsymbol{s})+\boldsymbol{b}_i))\boldsymbol{z}_{i+1}\big)J_{i-1}\Big) \\
&= \operatorname{tr}\Big(J_{i-1}^\top \boldsymbol{K}_i^\top \operatorname{diag}\big(\sigma''(\boldsymbol{K}_i\boldsymbol{u}_{i-1}+\boldsymbol{b}_i)\odot \boldsymbol{z}_{i+1}\big)\,\boldsymbol{K}_i J_{i-1}\Big) \\
&= \big(\sigma''(\boldsymbol{K}_i\boldsymbol{u}_{i-1}+\boldsymbol{b}_i)\odot \boldsymbol{z}_{i+1}\big)^\top\big((\boldsymbol{K}_i J_{i-1})\odot(\boldsymbol{K}_i J_{i-1})\big)\mathbf{1}.
\end{aligned}$$

Here, $J_{i-1} = \nabla_{\boldsymbol{s}}\boldsymbol{u}_{i-1}^\top \in \mathbb{R}^{m\times d}$ is a Jacobian matrix, which can be updated and over-written in the forward pass at a computational cost of $\mathcal{O}(m^2\cdot d)$ FLOPS. The J update follows:

$$\begin{aligned}
\nabla_{\boldsymbol{s}}\boldsymbol{u}_i^\top &= \nabla_{\boldsymbol{s}}\boldsymbol{u}_{i-1} + h\,\sigma'(\boldsymbol{K}_i\boldsymbol{u}_{i-1}+\boldsymbol{b}_i)\boldsymbol{K}_i^\top \nabla_{\boldsymbol{s}}\boldsymbol{u}_{i-1} \\
J &\leftarrow J + h\,\sigma'(\boldsymbol{K}_i\boldsymbol{u}_{i-1}+\boldsymbol{b}_i)\,\boldsymbol{K}_i^\top J
\end{aligned} \tag{30}$$

Since we parameterize the potential Φ instead of the $\mathbf{v}$, the Jacobian of the dynamics $\nabla\mathbf{v}$ is given by the Hessian of Φ in (2). We note that Hessians are *symmetric* matrices. We use the exact trace; however, if we wanted to use a trace estimate, a plethora of estimators perform better in accuracy and speed on symmetric matrices than on nonsymmetric matrices [5, 27, 53].

E Loss Metric

The testing loss metric C depends on the ℓ computation in (2), which is the integration of the trace along the computed trajectory $\boldsymbol{z}$. Different integration schemes have various error when integrating the trace [40, 55]. Too coarse of a time discretization can result in a low C value while sacrificing invertibility. Furthermore, a low C value does not imply good quality generation [52].

As a result, testing loss is unreliable for comparative evaluation of models' performances, so we use MMD.

Visualizations present the best evaluation of a flow's performance. We motivate this with a thorough comparison of OT-Flow against FFJORD (Fig. A2). Even though the testing losses are similar, the

Data Set	Model	# Param	Training Time (s)	Testing Loss	Inverse Error	MMD
Gaussian Mixture	OT-Flow	637	189	2.88	1.28e-8	6.38e-4
	FFJORD	9225	7882	2.85	7.22e-8	6.54e-4

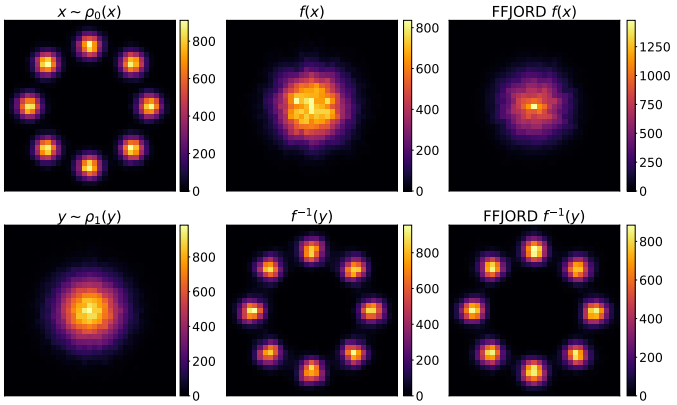


Figure A2: CNF performance heatmaps for the toy Gaussian mixture problem. (top row) 10^5 samples $\boldsymbol{x}$ from the pretended unknown ρ_0, the forward propagations of our flow $f(\boldsymbol{x})$ and the FFJORD flow. (bottom row) 10^5 samples $\boldsymbol{y}$ drawn from the known ρ_1, our model's generation $f^{-1}(\boldsymbol{y})$ using the inverse flow on normal samples and FFJORD's inverse flow on the same normal samples $\boldsymbol{y}$.

Discrete normalizing flow trained on POWER

d	**Testing Loss**	Inv Error	MMD
6	$\mathbf{-0.64}$	2.34e-3	1.94e-2

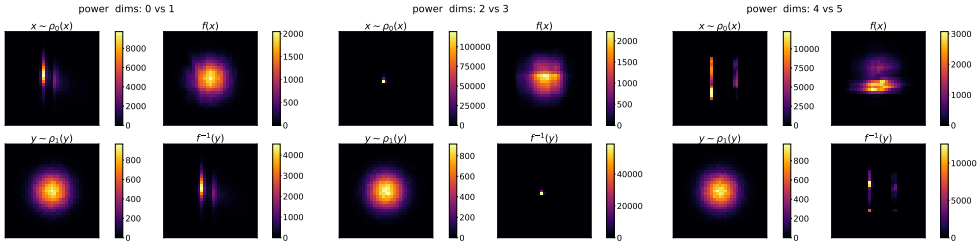


Figure A3: POWER density estimation for some discrete normalizing flow. By the testing loss metric, this model is considered very competitive. However, the model itself performs poorly, as clear in the visualization of the last two dimensions. The MMD shows that the generation is poor. The inverse error shows that the testing loss uses an integration scheme that is too coarse, as addressed in [40, 55].

FFJORD flow pushes too many points to the origin and does not map well to a Gaussian. We can see this flaw when using 10^5 samples.

In high-dimensions, visualizations become difficult, which is why reliance on a loss function is appealing. We visualize two-dimensional slices of these high-dimensional point clouds using binned heatmaps (App. F). We then get a sense for which dimensions are or are not mapping to ρ_1. For instance, we present an arbitrary finite normalizing flow trained on the POWER data set in which the testing loss looks competitive, but other metrics and the visualization demonstrate the model's flaws (Fig. A3). The last two dimensions show that the forward flow $f(\boldsymbol{x})$ noticeably differs from ρ_1 in these two dimensions. The associated MMD is poor for this model on the POWER data set (comparable MMDs in Tab. 2), and the high inverse error suggests that the integration is not trustworthy. However, the model achieves a testing loss of -0.64 which outperforms numerous state-of-the-art models (Tab. A1). Motivated by these demonstrations, we do not use the testing loss metric to evaluate flows.

Table A1: Testing Loss C comparison with other models.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
OT-Flow (Ours)	-0.31	-8.50*	17.46	10.52	-153.98
FFJORD trained by us	-0.42	-10.53	16.57	10.64	-142.91†
MADE [21]	3.08	-3.56	20.98	15.59	-148.85
RealNVP [14]	-0.17	-8.33	18.71	13.55	-153.28
Glow [29]	-0.17	-8.15	18.92	11.35	-155.07
MAF [41]	-0.24	-10.08	17.70	11.75	-155.69
NAF [26]	-0.62	-11.96	15.09	8.86	-157.73
UMNN [55]	-0.63	-10.89	13.99	9.67	-157.98

*This value is from a model trained using double precision and is different from the model reported in Tab. 2.

†Training manually terminated before convergence.

Table A2: Number of parameters comparison with discrete normalizing flows. FFJORD already reduced the parameterization of flows. We further reduce parameterization of CNFs, which includes a drastic reduction across all normalizing flows.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
OT-Flow (Ours)	17K	69K	72K	78K	297K
FFJORD [23]	43K	279K	547K	821K	6.7M
NAF [26]	414K	402K	9.27M	7.49M	36.8M
UMNN [55]	509K	815K	3.62M	3.46M	15.6M

Papamakarios et al. [41] cleaned and normalized the GAS data set used by other normalizing flow models (Tab. A1). However, after that preprocessing, some input values $\boldsymbol{x}$ still contain large values. Some models handle the large values by using normalization layers. While we can easily add normalization layers into OT-Flow, for simplicity, we further preprocess GAS. In particular, we scale all inputs by dividing by 5 before passing to the model. Alternatively, if we want to compare testing loss C with other methods, we can use double precision instead of scaling the inputs (Tab. A1). MMD and inverse error are similar for these two approaches.

F Visualizations of High-Dimensional Data Sets

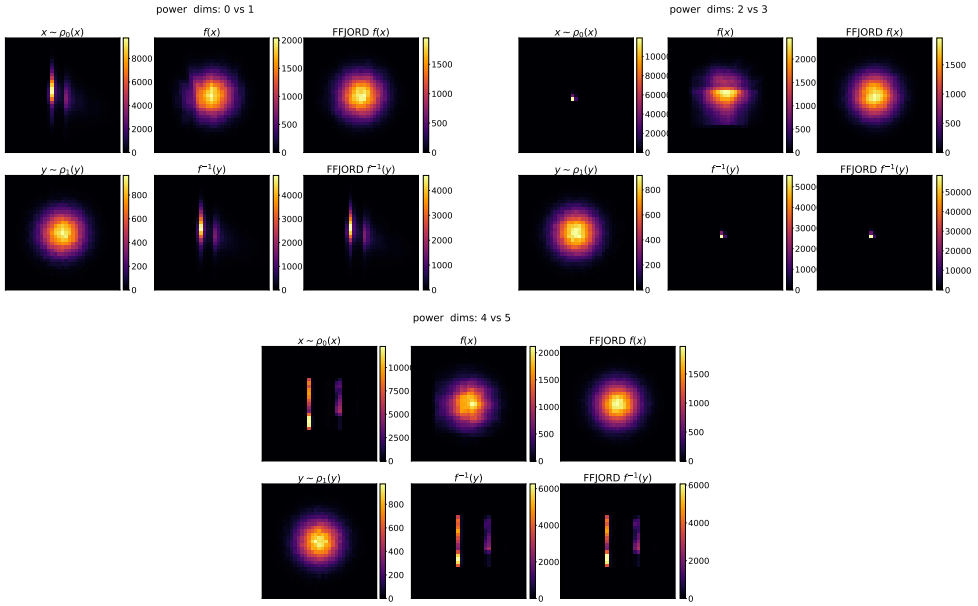


Figure A4: Model performance on POWER test data.

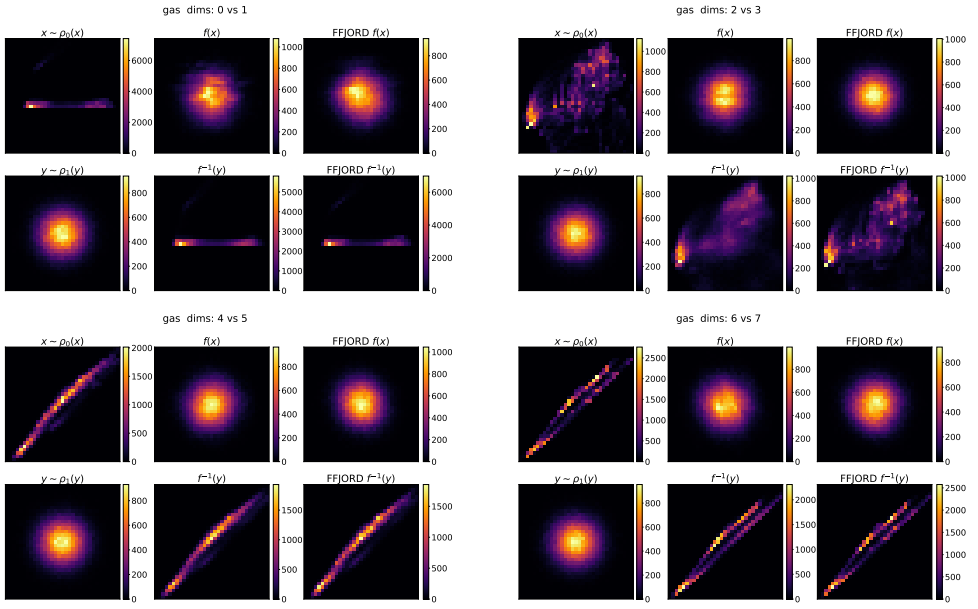


Figure A5: Model performance on GAS test data.

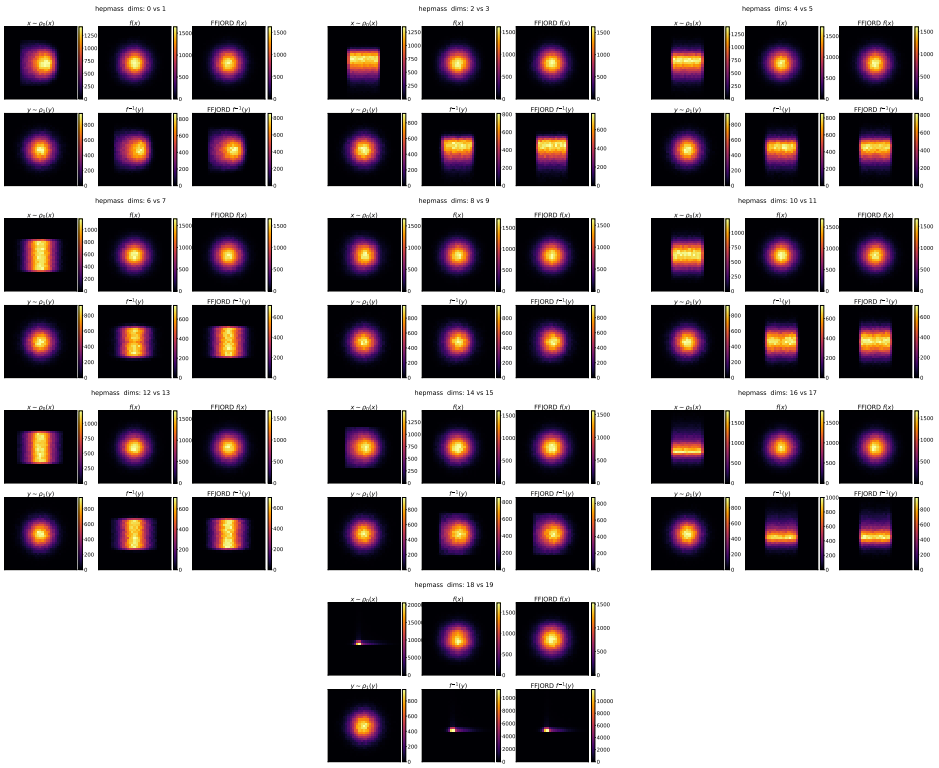


Figure A6: Model performance on the HEPMASS test data.

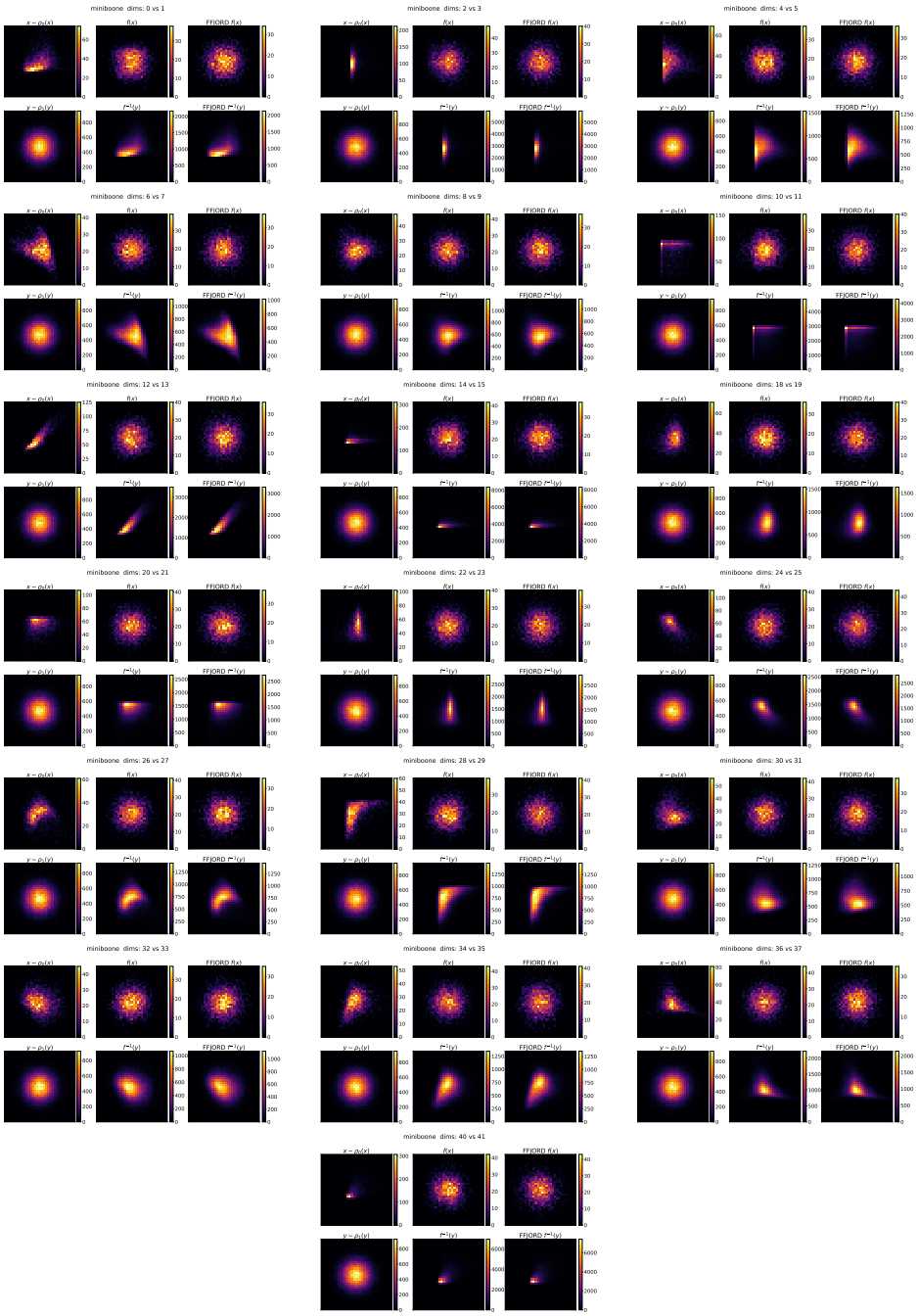


Figure A7: Other two-dimensional slices of the MINIBOONE density estimation to supplement Fig. 4.