TRADE: Transfer of Distributions between External Conditions with Normalizing Flows

Stefan Wahl[⋆] **Armand Rousselot**[⋆,†] **Felix Draxler** **Ullrich Köthe**
Computer Vision and Learning Lab, Heidelberg University
[⋆]Equal contribution
[†]Corresponding author: armand.rousselot@iwr.uni-heidelberg.de

Abstract

Modeling distributions that depend on external control parameters is a common scenario in diverse applications like molecular simulations, where system properties like temperature affect molecular configurations. Despite the relevance of these applications, existing solutions are unsatisfactory as they require severely restricted model architectures or rely on energy-based training, which is prone to instability. We introduce TRADE, which overcomes these limitations by formulating the learning process as a boundary value problem. By initially training the model for a specific condition using either i.i.d. samples or backward KL training, we establish a boundary distribution. We then propagate this information across other conditions using the gradient of the unnormalized density with respect to the external parameter. This formulation, akin to the principles of physics-informed neural networks, allows us to efficiently learn parameter-dependent distributions without restrictive assumptions. Experimentally, we demonstrate that TRADE achieves excellent results in a wide range of applications, ranging from Bayesian inference and molecular simulations to physical lattice models. Our code is available at `https://github.com/vislearn/trade`

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258. Copyright 2025 by the author(s).

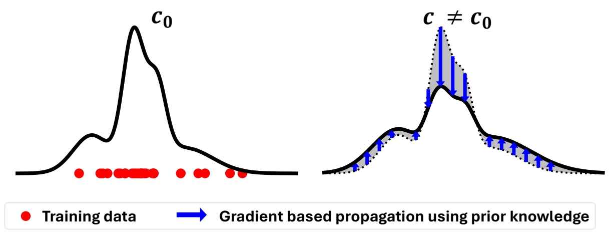

Figure 1: Our approach to train a conditional normalizing flow $p_\theta(x|c)$. **Left:** At $c = c_0$, the flow is trained using NLL. **Right:** By learning the gradient of the distribution with respect to c based on prior knowledge, the distribution learned at c_0 is propagated to other conditions $c \neq c_0$ without additional training data.

1 INTRODUCTION

Generative models have achieved impressive performance in scientific applications among many other fields (Noé et al., 2019; Butter and Plehn, 2022; Cranmer et al., 2020; Sanchez-Lengeling and Aspuru-Guzik, 2018). Oftentimes, such systems can depend on external control parameters, such as temperature governing the behavior of thermodynamic systems, coupling constants in physical models, or a tempered likelihood posterior in Bayesian inference (Wuttke et al., 2014; Friel and Pettitt, 2008; Pawlowski et al., 2017). A major challenge in such cases is acquiring training data for a complete range of control parameters, which can quickly become infeasible.

In this work, we focus on the common task of learning generative models in the case where we have access to the functional form of the unnormalized density, such as learning the distributions of equilibrium states in physical systems like molecules (Boltzmann distributions) or variational inference. We approach this problem by learning a single generative model that takes the external condition c as a parameter: $p_\theta(x|c) \approx p(x|c)$.

Several works have attempted to address this problem before. One approach applies architectural restrictions to allow one particular functional form of external parameter dependence (Dibak et al., 2022). However, this restriction has recently been shown to incur severe limitations in expressivity (Draxler et al., 2024b). Energy-based training has been proposed as another method (Schebek et al., 2024; Invernizzi et al., 2022; Wirnsberger et al., 2020), but can exhibit unfavorable properties like mode-seeking behavior (Minka et al., 2005; Felardos et al., 2023), which has also been shown to be a problem in practice (Wirnsberger et al., 2020) (see Appendix C for an example). Other works require data to be available at the target parameters (Wang et al., 2022b; Wirnsberger et al., 2020), which can become prohibitively expensive. We overcome these limitations: We allow learning arbitrary continuous dependencies of the target density on external parameters and train unconstrained architectures. Our central idea is to formulate the training of a conditional probability density as a boundary value problem: The boundary is learned on a fixed reference condition $c = c_0$, and then the resulting density is extrapolated using the known functional dependence on the condition underlying the problem. Our approach is summarized in Fig. 1.

In summary, we contribute:

- We introduce TRADE, a method for learning generative models with arbitrary continuous dependencies on external conditions. Learning is enabled by a novel boundary value problem formulation.
- TRADE uses unconstrained architectures, facilitating the application to more complex target distributions.
- TRADE can be trained data-free or with data only available at the boundary c_0, making it an efficient approach in cases where acquiring data for a full range of control parameters is infeasible.
- TRADE achieves excellent results in a wide range of experiments including Bayesian inference, molecular simulations and physical lattice models

2 RELATED WORK

A line of work which is closely related to ours are physics-informed neural networks (PINNs), which were first introduced by Raissi et al. (2019). The idea is to represent the solution of a PDE via a neural network, where the PDE is enforced via the loss function. The boundary values are usually given in the form of limited or noisy data. This approach has been successfully applied to a variety of scientific problems, including fluid dynamics, heat transfer, and material science (Cuomo et al., 2022). The works of Liu et al. (2023); Guo et al. (2022) combine PINNs with normalizing flows for solving SDEs like the Fokker-Planck equation. Cena et al. (2024) use a physics-informed loss to improve normalizing flow performance for detecting satellite power system faults. In our work, we similarly leverage a PDE constraint during training. However, our TRADE PDE is a function of the explicit dependence of the unnormalized distribution on an external control parameter. This is in contrast to standard PINNs, which typically enforce physical laws on the generated data itself. By embedding the control parameter directly into the PDE, TRADE is able to extrapolate to new control parameters even when only data from a single one is available.

A special case of distribution transfer is the generalization of a physical system across multiple thermodynamic states. The approach proposed by temperature steerable flows (TSF) (Dibak et al., 2022) learns a transformation of samples with constant Jacobian determinant, which makes temperature scaling of the learned data distribution equivalent to temperature scaling of the latent distribution. This method is restricted to temperature scaling and, as shown in Draxler et al. (2024b), such a transformation is severely limited in expressivity by design. This stands in contrast to TRADE, which introduces no additional restrictions on the model architecture. We showcase the effect of these architectural restrictions in Appendix C.

Conditional Boltzmann generators (BG) (Schebek et al., 2024) and learned replica exchange (LREX) (Invernizzi et al., 2022) map samples from one thermodynamic state to another by training the model with backward KL, which can lead to problems of mode seeking behavior (Minka et al., 2005; Felardos et al., 2023). Our method not only avoids these problems, but can also be trained knowing only the dependence of the energy function on the external parameter. This can even be possible in cases where computing the full ground truth energy of the system is infeasible, while the backward KL training objective requires this computation. Temperature-annealed Boltzmann generators Schopmans and Friederich (2025) tackle the task of training entirely data free at any target temperature. To avoid problems of backward KL training, which predominantly occur at low temperatures, they establish a boundary distribution at high temperatures using backward KL, then iteratively transfer it to the desired temperature by re-training with reweighted samples from the Boltzmann generator. As discussed in the next section, reweighting to low temperatures may lead to inefficient training due to low weights. Using TRADE to attempt to reduce the number of re-training steps could be an interesting future direction of research.

Learned Bennett acceptance ratio (LBAR) (Wirnsberger et al., 2020) learns to transfer distributions between two thermodynamic states to increase the efficiency of importance sampling, by using data from target thermodynamic state. Using data from multiple thermodynamic states, (Wang et al., 2022b) use diffusion models to interpolate between different temperatures and sample from the full range. Both of these methods require data at or around the target thermodynamic states. In contrast, TRADE has the ability to extrapolate from only a single thermodynamic state to others, by leveraging the information gained from the functional dependence of the unnormalized distribution on the control parameter.

3 BACKGROUND

Normalizing flows are a class of generative models that learn an invertible mapping $f_\theta : \mathbb{R}^d \mapsto \mathbb{R}^d$ mapping samples from the data distribution $p(x)$ to a tractable latent distribution $p_z(z)$, typically a Gaussian distribution (Kobyzev et al., 2020). New samples from the model can be obtained by passing samples of the latent distribution through the inverse function f_θ^{-1}. The model likelihood can be obtained by the change of variables formula

$$\log p_\theta(x) = \log p_z(f_\theta(x)) + \log \left| \frac{\partial f_\theta(x)}{\partial x} \right|. \quad (1)$$

Here, $\frac{\partial f_\theta(x)}{\partial x} \in \mathbb{R}^{d\times d}$ is the Jacobian matrix of f_θ and $|\cdot|$ denotes the absolute value of its determinant. In general, normalizing flows are trained by minimizing the negative log-likelihood (NLL) of the training data, which is equivalent to minimizing the (forward) KL divergence between data and model distribution

$$D_{\mathrm{KL}}(p(x)\|p_\theta(x)) = \mathbb{E}_{x\sim p(x)}[\log p(x) - \log p_\theta(x)]. \quad (2)$$

For typical neural networks, computing the Jacobian determinant during training can quickly become intractable. There are several ways this problem can be addressed. Some works define a surrogate objective that does not require the full Jacobian (Draxler et al., 2024a; Sorrenson et al., 2024). Other works construct neural networks using so-called coupling blocks (Dinh et al., 2014, 2016), whose structure makes the network Jacobian determinant and inverse directly tractable. We choose the latter approach for this work, as it has the advantage of allowing for a tractable computation of the likelihood during training.

Normalizing flows can also be trained in the absence of data (Felardos et al., 2023) if an (unnormalized) ground truth density $q(x) \propto p(x)$ is known, the two simplest methods being backward KL training and reweighting. Backward KL maximizes the likelihood of model samples under $p(x)$

$$D_{\mathrm{KL}}(p_\theta(x)\|p(x)) \propto \mathbb{E}_{p_\theta(x)}[\log p_\theta(x) - \log q(x)], \quad (3)$$

which can suffer from problems like mode seeking behavior (Minka et al., 2005; Felardos et al., 2023; Wirnsberger et al., 2020). Reweighting uses samples from another distribution $\hat{p}$ and reweights them to simulate forward KL training with $p(x)$

$$D_{\mathrm{KL}}(p(x)\|p_\theta(x)) \propto \mathbb{E}_{\hat{p}(x)}\left[\frac{q(x)}{\hat{p}(x)} \log\left(\frac{q(x)}{p_\theta(x)}\right)\right], \quad (4)$$

but can suffer from unstable or inefficient training resulting from high and low weights respectively (Agrawal et al., 2020). We illustrate failure cases of both backward KL training and reweighting in Appendix C.

4 METHOD

4.1 Conditional Distribution As Partial Differential Equation

We are interested in approximating conditional probability distributions:

$$p_\theta(x|c) \approx p(x|c) = q(x|c)\frac{1}{\int q(x|c)dx}, \quad (5)$$

where $q(x|c)$ is an unnormalized density and $p(x|c)$ its normalized variant.

We consider the case where (a) we can successfully train our model for some initial $p(x|c_0)$, for example using i.i.d. samples or via backward KL training (see Section 3), and (b) we have access to the dependence of the *unnormalized* $q(x|c)$ on c, as it is for example the case in physical systems where the temperature T regulates the relative probability of states in the form of the Boltzmann distributions:

$$q(x|T) = e^{-\frac{E(x)}{k_B T}}. \quad (6)$$

k_B is the Boltzmann constant.

This allows treating $p_\theta(x|c)$ as the solution to the following partial differential equation (PDE):

$$p_\theta(x|c_0) = p(x|c_0), \quad (7)$$

$$\frac{\partial}{\partial c} p_\theta(x|c) = \frac{\partial}{\partial c} p(x|c). \quad (8)$$

In Appendix A we show that this boundary value problem has a unique solution. Intuitively, Eq. (7) ensures that the correct distribution is learned at $c = c_0$. Equation (8) then propagates this information according to the gradient of the density. As noted in Section 3, the distribution at $p_\theta(x|c_0)$ can be learned from samples (forward KL) or using the unnormalized density $q(x|c_0)$. This ensures Eq. (7).

It is left to align the gradients of $p_\theta(x|c)$ with respect to c to the gradients of $p(x|c)$ to fulfill Eq. (8). Using the unnormalized density, we find:

$$\frac{\partial}{\partial c}\log p(x|c) = \frac{\partial}{\partial c}\log q(x|c) - \frac{\partial}{\partial c}\log\int q(x|c)dx. \quad (9)$$

We enable computing the latter term, the derivative of the normalization constant, via the following result:

Theorem 4.1. *Given an* unnormalized *density $q(x|c)$ that is differentiable in c, the derivative of the* normalized *density $p(x|c)$ with respect to c reads:*

$$\frac{\partial}{\partial c}\log p(x|c) = \frac{\partial}{\partial c}\log q(x|c) - \mathbb{E}_{p(x|c)}\left[\frac{\partial}{\partial c}\log q(x|c)\right] \quad (10)$$

This reformulation allows efficient training with Eq. (8) via the functional form of $q(x|c)$, which is tractable in many cases. For example, in the case of the Boltzmann distribution in Eq. (6):

$$\frac{\partial}{\partial T}\log q(x|T) = \frac{E(x)}{k_B T^2} \quad (11)$$

There are several choices to estimate the expectation value $\mathbb{E}_{p(x|c)}[\cdot]$, which we compare later in Section 5.1. The proof of Theorem 4.1 is straightforward:

Proof. Compute the derivative of the true normalized distribution with respect to the condition. We write the partition function as $Z(c) = \int q(x|c)dx$:

$$\frac{\partial}{\partial c}\log p(x|c) = \frac{\partial}{\partial c}\log q(x|c) - \frac{\partial}{\partial c}\log Z(c). \quad (12)$$

The only term left to consider is the derivative of the normalization as a function of the condition:

$$\begin{aligned}\frac{\partial}{\partial c}\log Z(c) &= \frac{1}{Z(c)}\frac{\partial}{\partial c}\int q(x|c)dx &(13)\\ &\overset{(*)}{=} \frac{1}{Z(c)}\int \frac{\partial}{\partial c}q(x|c)dx &(14)\\ &= \frac{1}{Z(c)}\int \frac{\partial}{\partial c}e^{\log q(x|c)}dx &(15)\\ &= \frac{1}{Z(c)}\int e^{\log q(x|c)}\frac{\partial}{\partial c}\log q(x|c)dx &(16)\\ &= \int p(x|c)\frac{\partial}{\partial c}\log q(x|c)dx. &(17)\end{aligned}$$

Combining Eqs. (12) and (17) yields the result. See Appendix A.2 for why we can swap integral and derivative in (∗). □

4.2 Training Objective

From the PDE in Eqs. (7) and (8), we construct the following loss terms for training. First, we consider some suitable objective $d(\cdot,\cdot)$ which learns the density at the boundary (see Section 3):

$$\mathcal{L}_{\text{boundary}} = d(p(x|c_0), p_\theta(x|c_0)). \quad (18)$$

Second, we add a loss that fixes the derivatives of the learned density to follow Eq. (8). We directly insert:

$$\begin{aligned}&\mathcal{L}_{\text{grad}}(c,x) = \\ &\left\|\tfrac{\partial}{\partial c}\log\left(\tfrac{p_\theta(x|c)}{q(x|c)}\right) + \mathbb{E}_{\tilde{x}\sim p(x|c)}\left[\tfrac{\partial}{\partial c}\log q(\tilde{x}|c)\right]\right\|^2.\end{aligned} \quad (19)$$

We evaluate the gradient loss $\mathcal{L}_{\text{grad}}(c,x)$ at points sampled from proposal distributions $p_{\text{grad}}(c)$ (see Section 4.3.2) and $p_{\text{grad}}(x|c)$ (see Section 4.3.3). To evaluate the expectation of the gradient w.r.t. c, we discuss possible options in Section 4.3.4.

Combining the two loss terms, we find the following differentiable optimization objective:

$$\mathcal{L} = \mathcal{L}_{\text{boundary}} + \lambda\,\mathbb{E}_{p_{\text{grad}}(c,x)}[\mathcal{L}_{\text{grad}}(c,x)]. \quad (20)$$

Inserting Eqs. (7) and (8) shows that this loss is indeed minimized for the true solution.

4.3 Practical Implementation

The formulation of the loss in Eq. (20) leaves several design choices open. We propose best practices in this section and perform an ablation in Section 5.1.

4.3.1 Discretization of the Condition

The first choice is to decide whether one should discretize the range of possible conditions $C = [c_{\min}, c_{\max}]$ into a grid $\tilde{C}$ or sample from C continuously. Discretization comes with the advantage of being able to save information from previous batches at each grid point and incorporate it into e.g. the definition of $p_{\text{grad}}(c)$ or a better estimation of $\mathbb{E}_{p(x|c)}\left[\frac{\partial}{\partial c}\log q(x|c)\right]$, while continuous sampling avoids discretization errors. In the following we provide design choices for both methods. For low- to medium-dimensional problems we find that both methods can perform equally well apart from a small discretization error (see Section 5.1). For high-dimensional problem where the ground-truth energy $q(x|c)$ is known, the discretized loss might yield better results.

4.3.2 Sampling of the Conditioning

The main idea behind our approach is that the flow learns the distribution explicitly only at the boundary $c = c_0$. The learned distribution is then propagated to $c \neq c_0$ through the gradient loss. Intuitively, this implies that the conditions near the boundary $c \approx c_0$ should be learned first, and the longer the training runs, the wider the interval of conditions should be.

Using the discretized grid of conditions $\tilde{C}$, we can apply a scheme inspired by the temporal causality weights proposed in Wang et al. (2022a). It samples conditions c weighted by the accumulated gradient loss from the reference point c_0:

$$p_{\text{grad}}(c) \propto \exp\left(-\epsilon \int_{c_0}^{c} \mathcal{L}_{\text{grad}}(\tilde{c}) d\tilde{c}\right) \tag{21}$$

$$\approx \exp\left(-\epsilon \sum_{c_i \in [c_0, c] \cap \tilde{C}} \mathcal{L}_{\text{grad}}(c_i)\right) \tag{22}$$

If $\mathcal{L}_{\text{grad}}(c')$ is large for some c', then the conditions further away from c_0 than c' are weighted down. We keep track of the loss values at the different conditions $\mathcal{L}_{\text{grad}}(c_i)$ during training using exponential averaging.

Without discretization, the integral in Eq. (21) is intractable. In this case we adapt a different schedule to sample c, based on the current training step t, which we describe in Appendix B.2.

4.3.3 Sampling of Evaluation Points

The next design decision is at which points x to evaluate the gradient loss. By Eq. (8), the gradient should follow the target distribution everywhere on the domain. In principle, we could therefore choose the proposal distribution $p_{\text{grad}}(x|c)$ arbitrarily as long as it captures the domain.

In practice, we find that choosing the model distribution $p_{\text{grad}}(x|c) = p_\theta(x|c)$ offers a good trade-off between exploration and exploitation. A small improvement can be achieved by perturbing the resulting samples with a small amount of Gaussian noise. In addition, we find that using prior knowledge about the sampling process, such as symmetries, can be beneficial.

4.3.4 Computation of Expectation Values

Theorem 4.1 allows to compute the derivative of the normalized density $\frac{\partial}{\partial c} p(x|c)$ via access to the derivative of the unnormalized density $\frac{\partial}{\partial c} q(x|c)$. However, it also involves its expectation over samples from $p(x|c)$:

$$\bar{\nabla}(c) := \mathbb{E}_{x \sim p(x|c)}\left[\frac{\partial}{\partial c} \log q(x|c)\right]. \tag{23}$$

For example, in Boltzmann distributions, this corresponds to the average energy at the evaluated temperature rescaled by $1/(k_B T^2)$.

To compute this expectation, we perform self-normalized importance sampling (SNIS) using N samples from a base distribution $p_{\text{base}}(x|c)$, together with the unnormalized density:

$$\bar{\nabla}(c) = \frac{\mathbb{E}_{x \sim p_{\text{base}}(x|c)}\left[\frac{q(x|c)}{p_{\text{base}}(x|c)} \frac{\partial}{\partial c} \log q(x|c)\right]}{\mathbb{E}_{x \sim p_{\text{base}}(x|c)}\left[\frac{q(x|c)}{p_{\text{base}}(x|c)}\right]} \tag{24}$$

$$\approx \frac{\sum_{i=1}^{N} \frac{q(x_i|c)}{p_{\text{base}}(x_i|c)} \frac{\partial}{\partial c} \log q(x_i|c)}{\sum_{i=1}^{N} \frac{q(x_i|c)}{p_{\text{base}}(x_i|c)}} \tag{25}$$

The numerator estimates the gradients, and the denominator correctly scales it with the estimated normalization constant of $q(x|c)$. Although SNIS performs well in our experiments and converges to the true expectation value, the estimator remains biased for finite N with a bias of $\mathcal{O}\left(\frac{1}{N}\right)$ (see, e.g., Cardoso et al. (2022)). In this scheme we can use any base distribution which covers the domain. In Section 5.1, we test different choices for $p_{\text{base}}(x|c)$ and again find that using the current model estimate $p_\theta(x|c)$ of the distribution at parameter c generally works best.

When using the discretized domain $\tilde{C}$, we find that evaluating the expectation values for every point on the grid in specified intervals and taking a running exponential average allows for using much larger sample sizes N and more accurate expectation value estimates at the same overall computational cost.

4.3.5 Energy-Free Training of TRADE

In some cases, only the functional form of the derivative $\frac{\partial q(x|c)}{\partial c}$ may be known, but $q(x|c)$ itself contains a term that is unknown (e.g. Section 5.2). In this case, we can often estimate this unknown term from the current model, via $q(x|c) \approx p_\theta(x|c)$.

For example, for performing temperature scaling of a Boltzmann distribution we are interested in the derivative in Eq. (11).

If $E(x)$ is unknown or computationally expensive, we can substitute $E(x) \approx -k_B T_0 \log p_\theta(x|T_0)$. This means we can train TRADE only based on samples, without access to the ground truth density. This is impossible with methods that rely on backward KL training.

5 EXPERIMENTS

For all experiments presented in this section, experimental details are provided in Appendix B.

5.1 Ablation Study: Multidimensional Wells

We perform an ablation study over the different design choices listed in the previous section and compare TRADE against training with backward KL, reweighted samples and TSF (Dibak et al., 2022). The task is to learn configurations x of a multidimensional well system in $d = 5$ dimensions, which consists of $2^5 = 32$ basins. Configurations follow a Boltzmann distribution (Eq. (6)) defined by the following energy function:

$$E(x) = \sum_{i=1}^{d} ax_i + bx_i^2 + cx_i^4. \tag{26}$$

Training data is given to all models at $T_0 = 1.0$, with the goal being to learn the distribution for a range $T \in [0.5, 1.0]$. The derivative required for training TRADE is given by Eq. (11).

For the discretization of T, we choose 15 grid points equidistantly in logarithmic space and use a periodically updated exponential running average for estimating the necessary expectation values as described in Section 4.3.4. While the backward KL model could in principle be trained entirely data-free, for the purpose of an equal comparison it is simultaneously trained with forward KL at $T = 1.0$ using the same training data that all other models receive (this can be regarded as a similar training procedure to Schebek et al. (2024)). In Table 1 we report the results of the all models in terms of NLL and (relative) effective sample size (ESS) (Kish, 1965).

The results provide a few guidelines for design choices in TRADE. Firstly, choosing $p_{\text{base}}(x|c) = p_\theta(x|c)$ seems to always be optimal. Secondly, when using a non-discretized loss, replacing $q(x|c)$ with an approximation by the model (as described in Section 4.3.5) barely impacts model performance, while it severely limits TRADE's performance when using a discretized loss. However, when using causality weights and the ground truth $q(x|c)$ the performance loss by discretizing is comparatively low. Compared to existing methods for temperature transfer TRADE outperforms all of them both in terms of NLL and ESS.

5.2 Tempered Bayesian Inference: Two Moons

We apply TRADE in a Bayesian inference setting (Cranmer et al., 2020), where the task is to learn the posterior distribution of the underlying parameters of a system ψ given observations y

$$p(\psi|y) \propto p(\psi)p(y|\psi), \tag{27}$$

where $p(\psi)$ is the prior and $p(y|\psi)$ is the likelihood of observation y given a parameter ψ. Tempered Bayesian inference learns a family of posteriors (Friel and Pettitt, 2008) which is defined as follows:

$$p(\psi|y, \beta) \propto p(\psi)p(y|\psi)^\beta. \tag{28}$$

The parameter β controls the trade-off between the prior and the data: $\beta = 1$ recovers the standard posterior, while $\beta > 1$ emphasizes the data and $\beta < 1$ emphasizes the prior. We train our model to perform tempered Bayesian inference controlled by β on the two moons task, introduced in Greenberg et al. (2019), a common benchmark for inference models. Observations y given parameters ψ are drawn as follows:

$$r \sim \mathcal{N}(0.1, 0.01^2) \tag{29}$$

$$\alpha \sim \mathcal{U}\big(-\frac{\pi}{2}, \frac{\pi}{2}\big) \tag{30}$$

$$y_1 = -\frac{|\psi_1 - \psi_2|}{\sqrt{2}} + r\cos(\alpha) + 0.25 \tag{31}$$

$$y_2 = -\frac{-\psi_1 + \psi_2}{\sqrt{2}} + r\sin(\alpha) \tag{32}$$

As opposed to Greenberg et al. (2019), we choose a non-uniform prior $p(\psi) = \mathcal{N}(0, 0.3^2)(\psi)$, making the task distinct from temperature scaling and preventing the application of TSF. The derivative needed for $\mathcal{L}_{\text{grad}}$ is the following:

$$\frac{\partial}{\partial\beta} \log q(\psi|y, \beta) \propto \log p(y|\psi) \tag{33}$$

While we have a closed-form solution of the likelihood $\log p(y|\psi)$ in this case, in many other applications this might not be available. As discussed in Section 4.3.5, a key advantage of TRADE is that it can be trained without access to the ground truth density, estimating $q(x|c)$ by the normalizing flow. We showcase this capability by only using the known prior $p(\psi)$ and the learned posterior $p_\theta(\psi|y)$ to estimate the likelihood $p(y|\psi) \approx \frac{p_\theta(\psi|y)}{p(\psi)}$ at training time.

In Fig. 2 we provide a comparison in terms of calibration of the estimated r (Eq. (29)) between a TRADE trained at $\beta \in [0.5, 2.0]$ compared to a model only using $\beta = 1.0$. At higher values of β, the tempered posterior of r contracts and the baseline model becomes underconfident, and vice-versa. In contrast, it is apparent that TRADE learns to generalize to different β values without incurring performance loss at $\beta = 1.0$.

5.3 Temperature Scaling of Alanine Dipeptide

We now demonstrate TRADE's ability to transfer data from molecular dynamics (MD) simulations between different temperatures. For this, we follow the setting from Dibak et al. (2022). The goal is to learn the

Table 1: Ablation study over different methods and design choices on a multidimensional-well system in $d = 5$ dimensions. We train models at temperature $T = 1.0$ and evaluate their ability to transfer the distribution to $T = 0.5$ in terms of NLL and ESS. Standard deviations are evaluated over 3 separate runs. TRADE (lower block) manages to achieve the best performance in comparison to several baseline methods (upper block). Gray rows mark configurations which were unstable.

Baselines				NLL T=0.5 ↓	NLL T=1.0 ↓	ESS T=0.5 ↑	ESS T=1.0 ↑
Forward KL training at $T = 1.0$ only				2.81 ± 0.0	4.37 ± 0.0	42.2 ± 0.2 %	98.8 ± 0.0 %
Backward KL training				2.29 ± 0.01	4.40 ± 0.0	84.1 ± 0.3 %	92.3 ± 0.2 %
Training with reweighted samples				2.23 ± 0.0	4.37 ± 0.0	91.3 ± 1.1 %	98.9 ± 0.0 %
Temperature-Steerable Flows (Dibak et al., 2022)				2.45 ± 0.02	4.45 ± 0.01	69.5 ± 1.7 %	81.7 ± 4.8 %
Discretize	$p_{\text{base}}(x\|c)$	Use ground truth $q(x\|c)$	Causality weights				
✓	$p_\theta(x\|c)$	✓	✗	11.3 ± 9.93	5.79 ± 0.26	0.0 ± 0.0 %	10.3 ± 2.3 %
✓	$p_\theta(x\|c)$	✗	✓	5.15 ± 1.15	4.58 ± 0.14	0.2 ± 0.3 %	63.7 ± 19.6 %
✗	$p_\theta(x\|c_0)$	✗		13.45 ± 4.56	9.2 ± 2.07	0.1 ± 0.1 %	0.6 ± 0.5 %
✗	$p(x\|c_0)$	✓		2.57 ± 0.04	4.63 ± 0.02	42.7 ± 0.02 %	59.9 ± 2.6 %
✗	$p(x\|c_0)$	✗		2.65 ± 0.03	4.48 ± 0.01	34.5 ± 3.5 %	78.4 ± 1.4 %
✗	$p_\theta(x\|c_0)$	✓		2.27 ± 0.0	4.38 ± 0.0	81.1 ± 0.4 %	97. ± 0.1 %
✗	$p_\theta(x\|c)$	✓		**2.22** ± 0.0	**4.36** ± 0.0	**96.7** ± 1.1 %	**99.3** ± 0.1 %
✗	$p_\theta(x\|c)$	✗		**2.22** ± 0.0	4.37 ± 0.0	**95.6** ± 2.3 %	**99.2** ± 0.1 %
✓	$p_\theta(x\|c)$	✓	✓	2.25 ± 0.0	4.37 ± 0.0	89.6 ± 0.5 %	97.9 ± 0.1 %

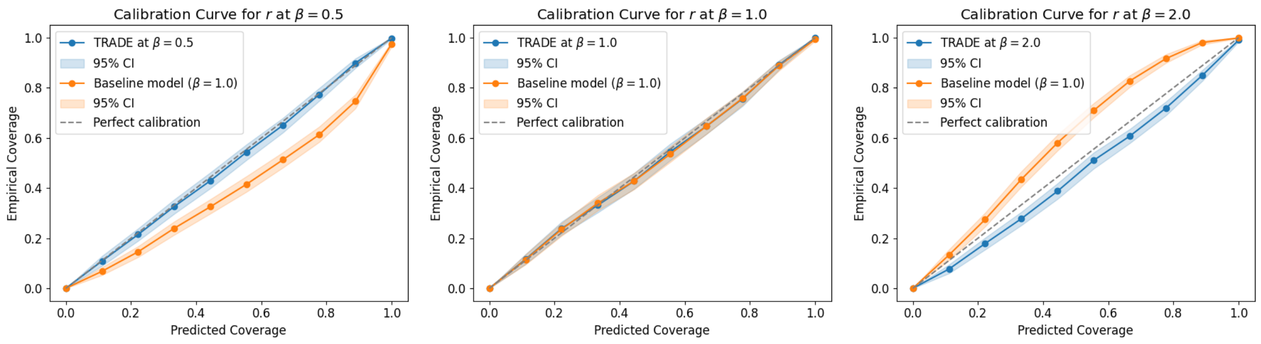

Figure 2: Calibration curves of the estimated r (Eq. (29)) at different likelihood powers β for the two moons dataset. We compare TRADE to a model trained only at $\beta = 1.0$ for $\beta \in \{0.5, 1, 2\}$. Our method successfully generalizes to different β while maintaining the same performance at $\beta = 1.0$. The model trained only at $\beta = 1.0$ fails to generalize to other values of β.

distribution of Alanine Dipeptide configurations x between different temperatures $T \in [300K, 600K]$. As before, the configurations follow a Boltzmann distribution with energy $E(x)$, with the required derivative being Eq. (11).

Table 2 compares TRADE to TSF (Dibak et al., 2022) and backward KL training. We train all models with MD simulation data at $600K$ from Dibak et al. (2022) and compare the resulting models at $600K$ and $300K$. For this we compute the negative log-likelihood (NLL) of the test data and the KL divergence between the 2D histograms (50×50 bins) of model samples and test samples of the ϕ, ψ angles of the molecule (see Dibak et al. (2022) figure 3 for an illustration). For each method we train two models, one based on affine coupling blocks (Dinh et al., 2016) and one based on the more expressive rational quadratic spline blocks (Durkan et al., 2019). To construct volume-preserving versions for TSF we use the architectures given in Dibak et al. (2022). For backward KL training we observe unstable training at the target temperature (especially in the affine model), while TRADE performs best with both architectures. It is worth noting that TSF drops in performance when using spline couplings, which we suspect is caused by a high approximation error of the temperature scaled spline. In Fig. 3 we show the best of three experiments for all spline models at $T = 300K$, which demonstrates TRADE's ability to extrapolate to lower temperatures accurately. Figures for affine models and other temperatures are provided in Appendix B.

Method	NLL $T = 300K \downarrow$	NLL $T = 600K \downarrow$	$D_{\text{hist}(\phi,\psi)}$ $T = 300K \downarrow$	$D_{\text{hist}(\phi,\psi)}$ $T = 600K \downarrow$
Backward KL Affine	-136.6067 ± 10.4910	-143.5167 ± 0.3188	0.1726 ± 0.0599	0.0438 ± 0.0038
TSF Affine (Dibak et al., 2022)	-159.1200 ± 0.1179	-144.3500 ± 0.0200	0.1707 ± 0.0503	**0.0331 ± 0.0040**
TRADE Affine (Ours)	**-159.8333 ± 0.0802**	**-144.4567 ± 0.0513**	**0.1402 ± 0.0290**	0.0353 ± 0.0141
Backward KL Spline	-160.7967 ± 0.4102	-145.1533 ± 0.0737	0.3982 ± 0.4716	0.0321 ± 0.0252
TSF Spline (Dibak et al., 2022)	-157.3600 ± 0.2364	-145.1400 ± 0.0000	0.1813 ± 0.0351	0.0200 ± 0.0019
TRADE Spline (Ours)	**-161.1633 ± 0.0153**	**-145.1900 ± 0.0000**	**0.0291 ± 0.0025**	**0.0077 ± 0.0002**

Table 2: Comparison of different temperature scaling methods trained on MD simulations of Alanine Dipeptide at 600K. We train all models with two different coupling architectures, affine coupling blocks (Dinh et al., 2016) and rational quadratic spline coupling blocks (Durkan et al., 2019), using the architecture presented in Dibak et al. (2022) to construct an approximately temperature scalable network for TSF. We evaluate models in terms of negative log-likelihood (NLL) and KL divergence between the 2D histograms of the ϕ, ψ of model samples and MD samples both at temperatures $T = 300K$ and $T = 600K$. Standard deviations are taken over three independent runs with the same hyperparameters.

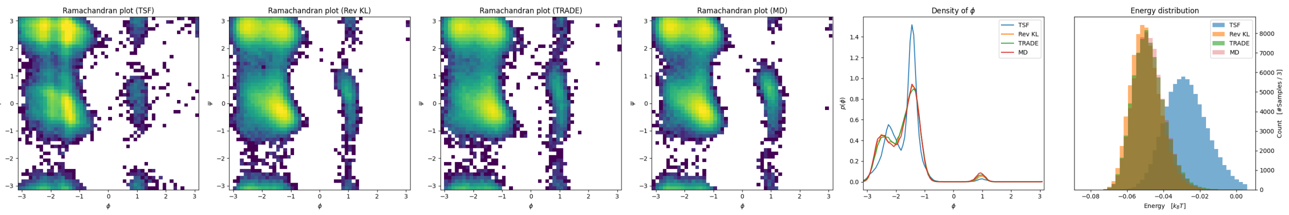

Figure 3: A comparison of TSF and backward KL training to TRADE trained on MD simulations of Alanine Dipeptide. Both models were trained at 600K and are evaluated at 300K. From left to right: Ramachandran plots of model samples (TSF, backward KL, TRADE) and molecular dynamics (MD), marginal density of the ϕ angle, ground truth energy of model and MD samples. For each model we plot the best result of three runs.

5.4 Varying External Parameters in a Scalar Field Theory

In this section we apply TRADE to a physical lattice model which is for example examined by Pawlowski et al. (2017). The distribution of states depends on two external parameters, the quartic coupling λ and the hopping parameter κ. The distribution of states is given by $p(\phi|\kappa, \lambda) \propto e^{-S(\phi,\lambda,\kappa)}$:

$$S(\phi, \lambda, \kappa) = \sum_x \left[-2\kappa \sum_{\mu=1}^{d} \phi_x \phi_{x+\hat{\mu}} + (1 - 2\lambda)\phi_x^2 + \lambda \phi_x^4 \right]. \quad (34)$$

The first sum runs over all positions on the grid, while the second sum runs over all dimensions of the grid accounting for nearest-neighbor interactions. An interesting property of this model is that it has a second order phase transition as κ is varied (see Pawlowski et al. (2017)).

We fix $\lambda = 0.02$ and train a normalizing flow conditioned on κ for $d = 2$ and a grid of size 8×8. We use training data left and right of the phase transition for maximum likelihood training and interpolate the distribution between these two κ by applying TRADE (in contrast to the previous experiments, where we only used data from one condition). As TSFs (Dibak et al., 2022) are not applicable to this problem due to the

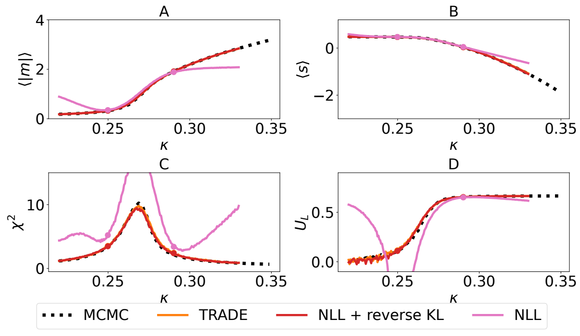

Figure 4: Physical observables for the scalar field theory. TRADE and a combination of NLL and energy-based training accurately follow the ground truth obtained using MCMC, while NLL alone is detrimental. **A:** Expected absolute magnetization per spin. **B:** Expected ground truth action per spin. **C:** Susceptibility. **D:** Binder cumulant. The values marked with dots represent the κ at which training data is available.

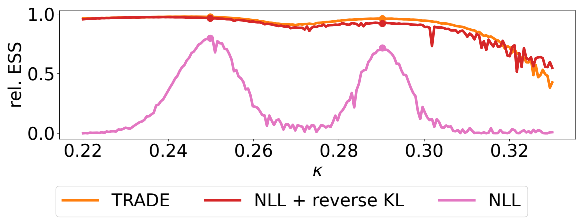

Figure 5: Relative ESS for the models trained for the scalar field theory. The values marked with dots represent the κ at which training data is available. TRADE outperforms the baseline in most of the examined range of κ and is also less fluctuating.

structure of Eq. (34), we use combined NLL and backward KL training as baseline. In addition, we train a model by only applying NLL training at the two κ where data is available.

Figure 4 shows some of the physical observables described in Pawlowski et al. (2017) based on the samples following the three training schemes. The model trained with NLL only fails to reproduce the MCMC simulation. TRADE and the baseline using NLL and backward KL both accurately follow the ground truth data obtained via MCMC. Figure 5 shows the corresponding ESS. Again, the NLL-only model falls short of the other models. TRADE and the NLL + reverse KL model perform similar at small κ, with a slight advantage for TRADE for intermediate values and for the baseline model at the highest values. Overall, the fluctuation in the ESS is smaller for TRADE than for the baseline model. Together, TRADE yields a performance gain compared to NLL training only and is competitive to the baseline. In Appendix B we evaluate the stability of the training schemes.

6 LIMITATIONS

While TRADE yields excellent results in all our evaluations, there are several points we hope can be improved in the future. Firstly, the evaluation of the expectation value in Eq. (19) may cause problems for high dimensional data sets or certain applications. Such problems may be extremely large batches, or extreme importance weights due to a bad alignment between the target and the proposal distribution. Secondly, TRADE requires an additional automatic differentiation step in each training iteration compared to other methods such as combined NLL and backward KL training, slightly increasing the computational requirements. However, none of these potential limitations posed issues in our experiments.

7 CONCLUSION

In this work we introduce TRADE, a novel method that learns to transfer distributions between different external parameters. TRADE first establishes a boundary distribution at a fixed external parameter c_0 by learning e.g. from i.i.d. samples or energy-based training. It then propagates this information to other external parameters using the gradient of the unnormalized density w.r.t. the external parameter. Previous methods in this field either had to rely on energy-based training, prone to mode collapse, or severely restrict model architecture. By leveraging the information gained from the functional form of the unnormalized density, TRADE avoids these limitations.

As a consequence, TRADE achieves excellent results in a diverse set of experiments, outperforming previous methods. We provide an extensive ablation study over design choices and benchmark against previous methods on a multidimensional well system. Our model can even be applied without access to the ground-truth unnormalized density, as demonstrated in a Bayesian inference setting. Finally, we demonstrate TRADE's ability to perform distribution transfer in complex settings such as learning configurations of Alanine Dipeptide and a scalar field theory lattice model.

Acknowledgments

We would like to thank Henrik Schopmans for providing crucial help with regards to training the Boltzmann Generators. This work is supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC-2181/1 - 390900948 (the Heidelberg STRUCTURES Cluster of Excellence). AR acknowledges funding from the Carl-Zeiss-Stiftung and by the German Federal Ministry of Education and Research (BMBF) (project EMUNE/031L0293A). UK thanks the Klaus Tschira Stiftung for their support via the SIMPLAIX project. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Agrawal, A., Sheldon, D. R., and Domke, J. (2020). Advances in black-box vi: Normalizing flows, importance weighting, and optimization. *Advances in Neural Information Processing Systems*, 33:17358–17369.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Ardizzone, L., Bungert, T., Draxler, F., Köthe, U., Kruse, J., Schmier, R., and Sorrenson, P. (2018-2022). Framework for Easily Invertible Architectures (FrEIA).

Bear, H. S. (2002). 7 - the integral of unbounded functions. In Bear, H., editor, *A Primer of Lebesgue Integration (Second Edition)*, pages 61–72. Academic Press, San Diego, second edition edition.

Butter, A. and Plehn, T. (2022). Generative networks for lhc events. In *Artificial intelligence for high energy physics*, pages 191–240. World Scientific.

Cardoso, G., Samsonov, S., Thin, A., Moulines, E., and Olsson, J. (2022). Br-snis: Bias reduced self-normalized importance sampling.

Cena, C., Albertin, U., Martini, M., Bucci, S., and Chiaberge, M. (2024). Physics-informed real nvp for satellite power system fault detection. *arXiv preprint arXiv:2405.17339*.

Cranmer, K., Brehmer, J., and Louppe, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. (2022). Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88.

Dibak, M., Klein, L., Krämer, A., and Noé, F. (2022). Temperature steerable flows and boltzmann generators. *Physical Review Research*, 4(4):L042005.

Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Draxler, F., Sorrenson, P., Zimmermann, L., Rousselot, A., and Köthe, U. (2024a). Free-form flows: Make any architecture a normalizing flow. In *International Conference on Artificial Intelligence and Statistics*, pages 2197–2205. PMLR.

Draxler, F., Wahl, S., Schnoerr, C., and Koethe, U. (2024b). On the universality of volume-preserving and coupling-based normalizing flows. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F., editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11613–11641. PMLR.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Neural spline flows.

Falcon, W. and The PyTorch Lightning team (2019). PyTorch Lightning.

Felardos, L., Hénin, J., and Charpiat, G. (2023). Designing losses for data-free training of normalizing flows on boltzmann distributions. *arXiv preprint arXiv:2301.05475*.

Friel, N. and Pettitt, A. N. (2008). Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 70(3):589–607.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Greenberg, D., Nonnenmacher, M., and Macke, J. (2019). Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*, pages 2404–2414. PMLR.

Guo, L., Wu, H., and Zhou, T. (2022). Normalizing field flows: Solving forward and inverse stochastic differential equations using physics-informed flow models. *Journal of Computational Physics*, 461:111202.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Huang, C.-W., Dinh, L., and Courville, A. (2020). Augmented normalizing flows: Bridging the gap between generative flows and latent variable models. *arXiv preprint arXiv:2002.07101*.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Invernizzi, M., Krämer, A., Clementi, C., and Noé, F. (2022). Skipping the replica exchange ladder with normalizing flows. *The Journal of Physical Chemistry Letters*, 13(50):11643–11649.

Jacobsen, J.-H., Smeulders, A., and Oyallon, E. (2018). i-revnet: Deep invertible networks.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Kish, L. (1965). Sampling organizations and groups of unequal sizes. *American sociological review*, pages 564–572.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. (2020). Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979.

Liu, F., Wu, F., and Zhang, X. (2023). Pinf: Continuous normalizing flows for physics-constrained deep learning. *arXiv preprint arXiv:2309.15139*.

Minka, T. et al. (2005). Divergence measures and message passing. Technical report, Technical report, Microsoft Research.

Noé, F., Olsson, S., Köhler, J., and Wu, H. (2019). Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147.

pandas development team, T. (2020). pandas-dev/pandas: Pandas.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*.

Pawlowski, J. M., Stamatescu, I.-O., and Ziegler, F. P. (2017). Cooling stochastic quantization with colored noise. *Physical Review D*, 96(11).

Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

Sanchez-Lengeling, B. and Aspuru-Guzik, A. (2018). Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365.

Schebek, M., Invernizzi, M., Noé, F., and Rogal, J. (2024). Efficient mapping of phase diagrams with conditional normalizing flows. *arXiv preprint arXiv:2406.12378*.

Schopmans, H. and Friederich, P. (2025). Temperature-annealed boltzmann generators. *arXiv preprint arXiv:2501.19077*.

Smith, L. N. and Topin, N. (2018). Super-convergence: Very fast training of neural networks using large learning rates.

Sorrenson, P., Draxler, F., Rousselot, A., Hummerich, S., Zimmermann, L., and Köthe, U. (2024). Lifting architectural constraints of injective flows. In *The Twelfth International Conference on Learning Representations*.

Sorrenson, P., Rother, C., and Köthe, U. (2020). Disentanglement by nonlinear ica with general incompressible-flow networks (gin).

Wang, S., Sankaran, S., and Perdikaris, P. (2022a). Respecting causality is all you need for training physics-informed neural networks.

Wang, S., Sankaran, S., Wang, H., and Perdikaris, P. (2023). An expert's guide to training physics-informed neural networks.

Wang, Y., Herron, L., and Tiwary, P. (2022b). From data to noise to data for mixing physics across temperatures with generative artificial intelligence. *Proceedings of the National Academy of Sciences*, 119(32):e2203656119.

Wes McKinney (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Wirnsberger, P., Ballard, A. J., Papamakarios, G., Abercrombie, S., Racanière, S., Pritzel, A., Jimenez Rezende, D., and Blundell, C. (2020). Targeted free energy estimation via learned mappings. *The Journal of Chemical Physics*, 153(14).

Wuttke, R., Hofmann, H., Nettels, D., Borgia, M. B., Mittal, J., Best, R. B., and Schuler, B. (2014). Temperature-dependent solvation modulates the dimensions of disordered proteins. *Proceedings of the National Academy of Sciences*, 111(14):5213–5218.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes, we provide this in Section 4.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes, we provide a comparison in Section 6.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. No, but we will publish code upon acceptance.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes, in Section 4.
   (b) Complete proofs of all theoretical results. Yes, in Section 4 and Appendix A.
   (c) Clear explanations of any assumptions. Yes.

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes, we list hyperparameters, used datasets and settings in appendix Appendix B and will provide code upon acceptance.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes, in Appendix B
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes, a description is provided with each experimental section and Appendix B.
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes, in Appendix B.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes, datasets are correctly attributed.
   (b) The license information of the assets, if applicable. Yes, licenses are provided at the dataset sources.
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable.
   (d) Information about consent from data providers/curators. Not Applicable, all dataset used are publicly available.
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable.

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable.
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable.
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable.

Supplementary Materials

A PROOFS

A.1 Uniqueness of the Solution of the PDE used in the TRADE Objective

In Section 4.1 we describe the PDE on which TRADE is based, as outlined in Eqs. (7) and (8). To ensure that TRADE converges to the true solution, i.e. the ground truth $p(x|c)$ it remains to be shown that the boundary value problem Eqs. (7) and (8) has a unique solution and that this solution corresponds to the ground truth. We begin by rewriting Eqs. (7) and (8) in terms of the logarithm and show that this boundary value problem has a unique solution which is given by $\log p(x|c)$. The motivation behind switching to the logarithmic form is that our practical implementation is based on this version, as it is easier to implement and provides more numerical stability.

Theorem A.1. *Let $p(x|c)$ be a conditional probability density with a continuous condition c which is differentiable with respect to c. Then, the initial value problem*

$$\log p_\theta(x|c_0) = \log p(x|c_0) \quad \forall x \tag{35}$$

$$\frac{\partial \log p_\theta(x|c)}{\partial c} = \frac{\partial \log p(x|c)}{\partial c} \quad \forall x, c \tag{36}$$

has a unique solution $\log p_\theta(x|c)$, which is equal to $\log p(x|c)$.

Proof. We assume that we have two solutions $\log p_\theta(x|c)$ and $\log \tilde{p}_\theta(x|c)$ of the boundary value problem Eqs. (35) and (36). Therefore, it follows that

$$\log p_\theta(x|c_0) = \log \tilde{p}_\theta(x|c_0) = \log p(x|c_0) \quad \forall x \tag{37}$$

$$\frac{\partial \log p_\theta(x|c)}{\partial c} = \frac{\partial \log \tilde{p}_\theta(x|c)}{\partial c} = \frac{\partial \log p(x|c)}{\partial c} \quad \forall x, c. \tag{38}$$

We are now interested in integrating Eq. (38) with respect to c. Without loss of generality, we consider the interval $[c_0, c]$. For $\log p_\theta(x|c)$ we find

$$\begin{aligned}\log p_\theta(x|c) &= \log p_\theta(x|c_0) + \int_{c_0}^{c} \frac{\partial \log p(x|c')}{\partial c} dc' &(39)\\ &\overset{(*)}{=} \log p(x|c_0) + \int_{c_0}^{c} \frac{\partial \log p(x|c')}{\partial c} dc'. &(40)\end{aligned}$$

In $(*)$, we used the fact that $\log p_\theta(x|c)$ satisfies the boundary condition Eq. (35). The right hand side of Eq. (40) only depends on the ground truth $\log p(x|c)$, and no longer on $\log p_\theta(x|c)$. Computing $\log \tilde{p}_\theta(x|c)$ analogously to Eq. (39) we define the difference function Eq. (41).

$$f(x, c) = \log p_\theta(x|c) - \log \tilde{p}_\theta(x|c) \tag{41}$$

By inserting Eq. (40) and the corresponding result for $\log \tilde{p}_\theta(x|c)$ into Eq. (41) we find

$$\begin{aligned} f(x,c) &= \log p(x|c_0) + \int_{c_0}^{c} \frac{\partial \log p(x|c')}{\partial c} dc' - \log p(x|c_0) - \int_{c_0}^{c} \frac{\partial \log p(x|c')}{\partial c} dc' && (42) \\ &= 0. && (43) \end{aligned}$$

Equation (43) directly shows that $\log p_\theta(x|c) \equiv \log \tilde{p}_\theta(x|c)$. Therefore, the boundary value problem Eqs. (35) and (36) has a unique solution.

To complete the proof, it is left to show, that the ground truth $\log p(x|c)$ is a solution of Eqs. (35) and (36). This follows trivially from the definition of the boundary value problem. Combining this with the uniqueness of the solution, we conclude that Eqs. (35) and (36) have a unique solution, which is indeed the ground truth $\log p(x|c)$. □

Theorem A.1 proves that the only solution to Eqs. (35) and (36) is the logarithm of ground truth $p(x|c)$. However, Eqs. (7) and (8) are formulated directly in terms of the distributions, rather than their logarithms. Nonetheless, the two formulations of the boundary value problem are equivalent, as demonstrated by the following theorem:

Theorem A.2. *Let $\log p_\theta(x|c)$ be a solution of the boundary value problem Eqs. (35) and (36). Then, $p_\theta(x|c) \coloneqq e^{\log p_\theta(x|c)}$ is a solution of the boundary value problem*

$$\begin{aligned} p_\theta(x|c_0) &= p(x|c_0) \quad \forall x && (44) \\ \frac{\partial p_\theta(x|c)}{\partial c} &= \frac{\partial p(x|c)}{\partial c} \quad \forall x, c. && (45) \end{aligned}$$

as introduced in Eqs. (7) and (8) in Section 4.1 of the main text.

Proof. To prove Theorem A.2, we insert $p_\theta(x|c) \coloneqq e^{\log p_\theta(x|c)}$ into Eqs. (44) and (45). Starting with Eq. (44) we find:

$$\begin{aligned} p_\theta(x|c_0) &= e^{\log p_\theta(x|c)} && (46) \\ &\overset{(*)}{=} e^{\log p(x|c)} && (47) \\ &= p(x|c_0). && (48) \end{aligned}$$

For Eq. (45) we compute:

$$\begin{aligned} \frac{\partial p_\theta(x|c)}{\partial c} &= \frac{\partial e^{\log p_\theta(x|c)}}{\partial c} && (49) \\ &= e^{\log p_\theta(x|c)} \frac{\partial \log p_\theta(x|c)}{\partial c} && (50) \\ &\overset{(*)}{=} e^{\log p(x|c)} \frac{\partial \log p(x|c)}{\partial c} && (51) \\ &= \frac{\partial e^{\log p(x|c)}}{\partial c} && (52) \\ &= \frac{\partial p(x|c)}{\partial c} && (53) \end{aligned}$$

In $(*)$, we use the fact that $\log p_\theta(x|c)$ is a solution of the boundary value problem Eqs. (35) and (36), which implies $\log p_\theta(x|c) \equiv \log p(x|c)$. □

Remark A.1. *The uniqueness of the solution of the boundary value problem Eqs.* (44) *and* (45) *can be shown analogously to the proof of Theorem A.1. By combining (a) the fact that* $\log p(x|c)$ *is the unique solution of Eqs.* (35) *and* (36) *and (b) the result of Theorem A.2 with this uniqueness of the solution of Eqs.* (44) *and* (45)*, we conclude that* $p(x|c)$ *is the unique solution to Eqs.* (44) *and* (45)*.*

A.2 Interchanging Derivative and Integration

Theorem 4.1 in the main text is crucial to TRADE, as it allows for computing the derivative of the logarithm of the normalized density with respect to the condition in terms of the known unnormalized density. In the proof of Theorem 4.1, the derivative with respect to the condition is interchanged with the integral over the support of the density. In this section, we provide a theoretical justification for this interchange.

Theorem A.3. *Let* $f(x,c)$ *be a integrable function with a continuous condition* c*. We assume that* $f(x,c)$ *is differential with respect to* c*. For convenience, we define* $I(c) := \int f(x,c)dx$*. Furthermore, we assume that* $\frac{\partial I(c)}{\partial c}$ *exists. If one can construct a function* $g(x,c)$ *such that* $|g_n(x,c)| \leq g(x,c)\ \forall x$ *with* $g_n(x,c) := \frac{f(x,c+h_n)-f(x,c)}{h_n}$*,* $h_n \to 0$ *as* $n \to \infty$*, then*

$$\frac{\partial}{\partial c}I(c) = \int \frac{\partial}{\partial c}f(x,c)dx \tag{54}$$

holds true.

Before we prove Theorem A.3, we first cite a result concerning the interchange of limits and integrals:

Theorem A.4 (The Lebesgue Dominated Convergence Theorem, Proposition 6 of Bear (2002))**.** *If* $\{f_n\}$ *is a sequence of measurable functions and* $f_n \to f$ *a.e. on a set* S *and there is an integrable function* g *on* S *such that* $|f_n| \leq g$ *for all* n*, then* f *is integrable and* $\int_S f_n \to \int_S f$*.*

We are now ready to prove Theorem A.3:

Proof. Since $\frac{\partial I(c)}{\partial c}$ is assumed to exist, we can rewrite it using its definition:

$$\frac{\partial I(c)}{\partial c} = \lim_{n\to\infty} \frac{I(c+h_n)-I(c)}{h_n} \tag{55}$$

Here, $\{h_n\}_{n\in\mathbb{N}}$ is a sequence in $\mathbb{R}$ with $h_n \to 0$ as $n \to \infty$. By exploiting the linearity of the integral, we find

$$\begin{aligned}\frac{\partial I(c)}{\partial c} &= \lim_{n\to\infty} \frac{I(c+h_n)-I(c)}{h_n} && (56)\\ &= \lim_{n\to\infty} \int \frac{f(x,c+h_n)-f(x,c)}{h_n}dx && (57)\\ &= \lim_{n\to\infty} \int g_n(x,c)dx && (58)\end{aligned}$$

where $g_n(x,c) := \frac{f(x,c+h_n)-f(x,c)}{h_n}$.

Since $\frac{\partial f(x,c)}{\partial c}$ is assumed to exist, by the definition of the derivative, we know that $g_n(x,c) \to \frac{\partial f(x,c)}{\partial c}$ as $n \to \infty$. If we can construct a function $g(x,c)$ such that $|g_n(x,c)| \leq g(x,c)\ \forall x$, then applying Theorem A.4 concludes the proof. $\square$

The proof of Theorem A.3 relies on the existence of a function g that bounds $|g_n(x,\beta)|$. In Example A.1, we construct such an upper bound for the case of temperature scaling, as for applied in Sections 5.1 and 5.3 of the main text.

Example A.1. *In the case of temperature scaling, the function $f(x,c)$ used in Theorem A.3 can be identified with the unnormalized Boltzmann distribution $q(x|T) = e^{-\frac{E(x)}{k_B T}}$. For the sake of notation, we switch to the generalized inverse temperature $\beta \coloneqq \frac{1}{k_B T}$, leading to $q(x|\beta) = e^{-\beta E(x)}$. We assume that the expectation value of the energy $\mathbb{E}_{p(x|\beta)}[E(x)]$ exists. This is a reasonable assumption, as the expected energy is a common observable in many applications. We can now define $g_n(x,\beta)$ as follows:*

$$g_n(x,\beta) = \frac{e^{-(\beta+h_n)\cdot E(x)} - e^{-\beta\cdot E(x)}}{h_n} \tag{59}$$

$$= e^{-\beta\cdot E(x)}\frac{e^{-h_n\cdot E(x)} - 1}{h_n}. \tag{60}$$

Let's assume without loss of generality, that $E(x) > 0$ and $h_n > 0$. Under this assumption, we find

$$|g_n(x,\beta)| = e^{-\beta\cdot E(x)}\frac{1 - e^{-h_n\cdot E(x)}}{h_n}. \tag{61}$$

To show that $g(x,\beta) = E(x)\cdot e^{-\beta\cdot E(x)}$ is an upper bound for $|g_n(x,\beta)|$, it is sufficient to prove that $\frac{1-e^{-h_n\cdot E(x)}}{h_n} \le E(x)$. The function $\frac{1-e^{-h_n\cdot E(x)}}{h_n} \le E(x)$ is strictly monotonically decreasing in h_n for $h_n > 0$. Therefore, the global maximum has to be attained for $h_n \to 0$. Using L'Hôspitals rule, we find Eq. (63).

$$\lim_{n\to\infty}\frac{1 - e^{-h_n\cdot E(x)}}{h_n} = \lim_{n\to\infty}\frac{E(x)\cdot e^{-h_n\cdot E(x)}}{1} \tag{62}$$

$$= E(x) \tag{63}$$

In conclusion, for temperature scaling, $g(x,\beta) = E(x)\cdot e^{-\beta E(x)}$ serves as a valid upper bound for $|g_n(x,\beta)|$ used in the proof of Theorem A.3.

B EXPERIMENTAL DETAILS

B.1 Loss Balancing

In the full TRADE objective in Eq. (20), the two loss contributions $\mathcal{L}_{\text{boundary}}$ and $\mathcal{L}_{\text{grad}}$, are balanced by a hyperparameter λ. To ensure that both contributions are minimized equally during each update step, we do not use a fixed λ throughout training. Instead, λ is computed adaptively. This computation scheme is based on the loss balancing scheme described in Wang et al. (2023).

$$\hat{\lambda}_{\text{boundary}} = \frac{||\nabla_\theta \mathcal{L}_{\text{boundary}}(\theta)|| + ||\nabla_\theta \mathcal{L}_{\text{grad}}(\theta)||}{||\nabla_\theta \mathcal{L}_{\text{boundary}}(\theta)||} \tag{64}$$

$$\hat{\lambda}_{\text{grad}} = \frac{||\nabla_\theta \mathcal{L}_{\text{boundary}}(\theta)|| + ||\nabla_\theta \mathcal{L}_{\text{grad}}(\theta)||}{||\nabla_\theta \mathcal{L}_{\text{grad}}(\theta)||} \tag{65}$$

To obtain λ used in Eq. (20) we define $\lambda \coloneqq \frac{\hat{\lambda}_{\text{grad}}}{\hat{\lambda}_{\text{boundary}}}$. By applying this weighting scheme, the magnitudes of the gradients of the two (weighted) contributions with respect to the model parameter θ are equal in each iteration. This ensures that the optimization of the objective is not dominated by a single contribution, but instead considers the full objective.

In practice, following Wang et al. (2023),we update $\hat{\lambda}_{\text{boundary}}$ and $\hat{\lambda}_{\text{grad}}$ every n training iterations. This reduces the computational overhead associated with the additional automated differentiation steps required to compute $\hat{\lambda}_{\text{boundary}}$ and $\hat{\lambda}_{\text{grad}}$. Additionally, we smooth $\hat{\lambda}_{\text{boundary}}$ and $\hat{\lambda}_{\text{grad}}$ by applying exponential averaging controlled by the parameter α_{balance}.

In our implementation, $\hat{\lambda}_{\text{boundary}}$ is computed based on the batch of training data used in the update step before the weights are updated. $\hat{\lambda}_{\text{grad}}$ is computed through a separate, data free calculation of $\mathcal{L}_{\text{grad}}$, using the current state of the training routine (i.e. causality weights and stored expectation values, as described in Section 4.3).

B.2 Sampling of the Condition for the Continuous Gradient Loss

As described in Section 4.3.2, we can define $p_{\text{grad}}(c)$ based on causality weights only if we discretize the domain of C. In the alternative case, we define different sampling schedule from $p_{\text{grad}}(c)$, which starts with the distribution concentrated around c_0 and progressively moves outward based on the current training step t. Given hyperparameters $s_{\min}, s_{\max}$, which determine the skew of the distribution towards c_0 at the start and end, respectively, we sample c as follows:

$$r \sim \mathcal{U}_{[0,1]} \tag{66}$$

$$w \sim \mathcal{B}\left(\frac{\log c_{\max} - \log c_0}{\log c_{\max} - \log c_{\min}}\right) \tag{67}$$

$$\zeta := \exp\left(\log s_{\min} + \frac{t}{t_{\max}}\left(\log s_{\max} - \log s_{\min}\right)\right) \tag{68}$$

$$c = \begin{cases} \exp\left(\log c_0 + (1 - r^{\zeta})(\log c_{\max} - \log c_0)\right) & \text{if } w = 1 \\ \exp\left(\log c_0 + (1 - r^{\zeta})(\log c_{\min} - \log c_0)\right) & \text{if } w = 0 \end{cases} \tag{69}$$

The parameter w chooses whether to move upward or downward from c_0 and r^{ζ} controls the magnitude of the shift. The decay/growth behavior of r^{ζ} is governed by $s_{\min}$ and $s_{\max}$. For $\zeta = 0$ we recover a δ-distribution around $c = c_0$; $\zeta = 1$ results in a log-uniform distribution and $\zeta \to \infty$ results in δ-distributions around $c_{\min}$ and $c_{\max}$. In general, we find that $s_{\min} = 0.01, s_{\max} = 1.5$ are good choices.

B.3 Ablation Study: Multidimensional Wells

We generate data according to the energy given in Eq. (26) using 32 MCMC chains, each initialized in one of the basins. A step size of 500 is used for $T = 1.0$, creating 300,000 data points while a step size of 1000 is used for $T = 0.5$ to create 30,000 data points. All models use RQ-spline coupling blocks (Durkan et al., 2019) without augmentation dimensions, except for TSF, which employs 25 volume-preserving affine coupling blocks with five augmentation dimensions. We find that increasing the learning rate of TSF to 1×10^{-3} improves its performance. Table 3 lists the remaining hyperparameters.

B.4 Tempered Bayesian Inference: Two Moons

We train models on the task introduced by Greenberg et al. (2019), but modify the prior distribution to $p(\psi) = \mathcal{N}(0, 0.3^2)$, as described in the main text. For training, we need to estimate

$$\frac{\partial \log p_{\beta}(\psi|y)}{\partial \beta} \propto \log p(y|\psi), \tag{70}$$

which we approximate as $\log p(y|\psi) \approx \log p_{\theta}(\psi|y) - \log p(\psi)$.

For evaluation, we need to sample from $\log p(\psi|y, \beta)$ in order to compute the calibration curves. This is equivalent to sampling from $p(\psi)$ and generating observations from $p(y|\psi)^{\beta}$. We can approximate $p(y|\psi)^{\beta} \propto p(r)^{\beta} p(\alpha)^{\beta}$, which is tractable since $p(r)^{\beta} = \mathcal{N}(0.1, \frac{0.01^2}{\beta})(r)$ and $p(\alpha)^{\beta} = p(\alpha)$.

The likelihood $p(y|\psi)$ forms a half-moon distribution, with the width being controlled by r. As β becomes too small, the variance of r increases, and the approximation $p(y|\psi)^{\beta} \propto p(r)^{\beta} p(\alpha)^{\beta}$ may become inaccurate. However, in the considered regime of $\beta \in [0.5, 2.0]$, the approximation is sufficiently accurate.

We compute the calibration by drawing 300,000 pairs y, ψ as described above, then for each y, we draw 300 model samples of $\tilde{\psi}$ and compute the corresponding $\tilde{r}$. Starting from the median of the $\tilde{r}$, we move outward until we reach the ground truth r, and report the coverage obtained in this way in Fig. 2. The 95% confidence intervals are obtained via bootstrap sampling.

Table 3: Hyperparameters for the multidimensional well experiment. Parameters exclusively used by TRADE are highlighted in gray.

Hyperparameter	Value
Train, val, test split	0.8, 0.1, 0.1
Coupling blocks	15
Coupling type	RQ Splines (Durkan et al., 2019)
Hidden dimensions	[256, 256]
Latent distribution	normal
Activation	SiLU
Learning rate	2×10^{-4}
Weight decay	0.0
Optimizer	Adam
Gradient clipping	3.0
Dequantization noise	0.0
Batch size	512
Training steps	10,000
Lr scheduler	one-cycle lr (Smith and Topin, 2018)
$\lambda_{\mathcal{L}_{\text{grad}}}$	1.0
Start $\mathcal{L}_{\text{grad}}$ at step	2,000
Grid points	15
ϵ causality weights	0.9
Expectation value update interval	100 steps
Expectation value update samples	5,000
$p_{\text{grad}}(x\|c)$	$p_\theta(x\|c) + \mathcal{N}(0, 0.0003^2)$

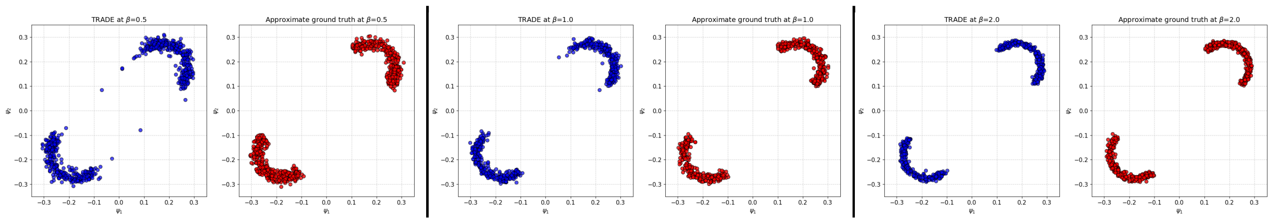

Figure 6: Learned (blue) vs. estimated (red) posterior samples of ψ at different β values for the observation $r, \alpha = 0$, by TRADE in the two moons experiment.

Hyperparameters are listed in Table 4 and in Fig. 6, we provide samples from the learned posterior distribution at different β for $r, \alpha = 0$.

B.5 Temperature Scaling of Alanine Dipeptide

We follow the setting of Dibak et al. (2022) to train a normalizing flow that learns the distributions of Alanine Dipeptide configurations at $T \in [300K, 600K]$, using their publicly available dataset. The configurations are first transformed into internal coordinates (torsions, angles, bonds), using a similar coordinate transform to Noé et al. (2019). All three methods are tested for affine coupling blocks (Dinh et al., 2016) and rational quadratic spline coupling blocks Durkan et al. (2019). Affine coupling blocks are made volume-preserving for TSF by normalizing the logarithmic scaling coefficients s of each block to 0. For a volume-preserving spline coupling architecture we use the architecture proposed in figure 1 of Dibak et al. (2022) in combination with the approximation for temperature steerable rational quadratic splines the authors present in appendix B. The latent distributions for affine models are scaled according to the temperature at which they are evaluated. The subnetworks of each block are single residual blocks, with an input and output layer scaling to and from the hidden size. The remaining hyperparameters are listed in Table 5. Additionally, we provide a visualizations of results of all architectures at

Table 4: Hyperparameters for the two moons experiment. Parameters exclusively used by TRADE are highlighted in gray.

Hyperparameter	Value
Train, val, test split	0.8, 0.1, 0.1
Coupling blocks	15
Coupling type	RQ Splines (Durkan et al., 2019)
Hidden dimensions	[128, 128]
Latent distribution	normal
Activation	SiLU
Learning rate	4×10^{-4}
Weight decay	0.0
Optimizer	Adam
Gradient clipping	3.0
Dequantization noise	3×10^{-4}
Batch size	512
Training steps	10,000
Lr scheduler	one-cycle lr (Smith and Topin, 2018)
$\lambda_{\mathcal{L}_{\text{grad}}}$	0.1
Start $\mathcal{L}_{\text{grad}}$ at step	2,000
Discretize	no
$p_{\text{base}}(x\|c)$	$p_\theta(x\|c)$
$p_{\text{grad}}(x\|c)$	$p_\theta(x\|c) + \mathcal{N}(0, 0.0003^2)$

$T = 600K$ and $T = 300K$ in Fig. 7. We performed a preliminary hyperparameter search using optuna (Akiba et al., 2019), then finetuned them for each model by hand.

B.6 Varying External Parameters in a Scalar Field Theory

B.6.1 Physical Observables

In Fig. 4 we visualize some physical observables taken from Pawlowski et al. (2017). The expected absolute magnetization per spin (subplot A) is defined by Eq. (72). The expected ground truth action per spin (subplot B) is given by Eq. (73), where $S(\phi, \kappa)$ corresponds to Eq. (34) with $\lambda = 0.02$. Equation (74) defines the susceptibility (subplot C), and finally, Eq. (75) defines the Binder cumulant (subplot D). Here, N represents the number of lattice points.

$$m(\phi) = \frac{1}{N}\left|\sum_x \phi_x\right| \tag{71}$$

$$\langle m(\kappa)\rangle = \mathbb{E}_{p_\theta(x|\kappa)}\left[m(\phi)\right] \tag{72}$$

$$\langle s(\kappa)\rangle = \frac{1}{N}\,\mathbb{E}_{p_\theta(x|\kappa)}\left[S(\phi, \kappa)\right] \tag{73}$$

$$\chi^2(\kappa) = N\left(\mathbb{E}_{p_\theta(x|\kappa)}\left[m(\phi)^2\right] - \mathbb{E}_{p_\theta(x|\kappa)}\left[m(\phi)\right]^2\right) \tag{74}$$

$$U_L(\kappa) = 1 - \frac{1}{3}\frac{\mathbb{E}_{p_\theta(x|\kappa)}\left[m(\phi)^4\right]}{\mathbb{E}_{p_\theta(x|\kappa)}\left[m(\phi)^2\right]^2} \tag{75}$$

In our experiments, the expectation values in Eq. (72) to Eq. (75) are approximated by the averaging over 10,000 samples drawn from the learned distributions.

Table 5: Hyperparameters for TRADE and TSF for the temperature scaling of Alanine Dipeptide. Parameters exclusively used by TRADE are highlighted in gray. Whenever different methods use different hyperparameters they are listed in the order TRADE/TSF/Backward KL

Hyperparameter	Affine Models	Spline Models
Train, val, test split	0.8, 0.05, 0.15	0.8, 0.05, 0.15
Coupling blocks	20	18
Hidden dimensions	[128, 128, 128]	[256, 256]
Augmentation dimensions (Huang et al., 2020)	60	0
Latent distribution	normal	uniform + truncated normal
Activation	SiLU	SiLU
Learning rate	3×10^{-4}	1×10^{-4}
Weight decay	$1 \times 10^{-6}/1 \times 10^{-5}/1 \times 10^{-5}$	3.7×10^{-6}
Optimizer	Adam	Adam
Gradient clipping	3.0	3.0
Dequantization noise	3×10^{-4}	3×10^{-4}
Batch size	1024	289
Training steps	58k	83k
Lr scheduler	one-cycle lr (Smith and Topin, 2018)	one-cycle lr
$\lambda_{\mathcal{L}_{\text{grad}}}$	0.1	0.23
Start $\mathcal{L}_{\text{grad}}$ at step	23k	25k
Discretize	no	yes
Grid points	-	25
ϵ causality weights	-	0.12
Expectation value update interval	-	1,000 steps
Expectation value update samples	-	5,000
$p_{\text{base}}(x\|c)$	$p_\theta(x\|c)$	$p_\theta(x\|c)$
$p_{\text{grad}}(x\|c)$	$p_\theta(x\|c) + \mathcal{N}(0, 0.0003^2)$	$p_\theta(x\|c) + \mathcal{N}(0, 0.0003^2)$
Use ground truth $q(x\|c)$	yes	yes

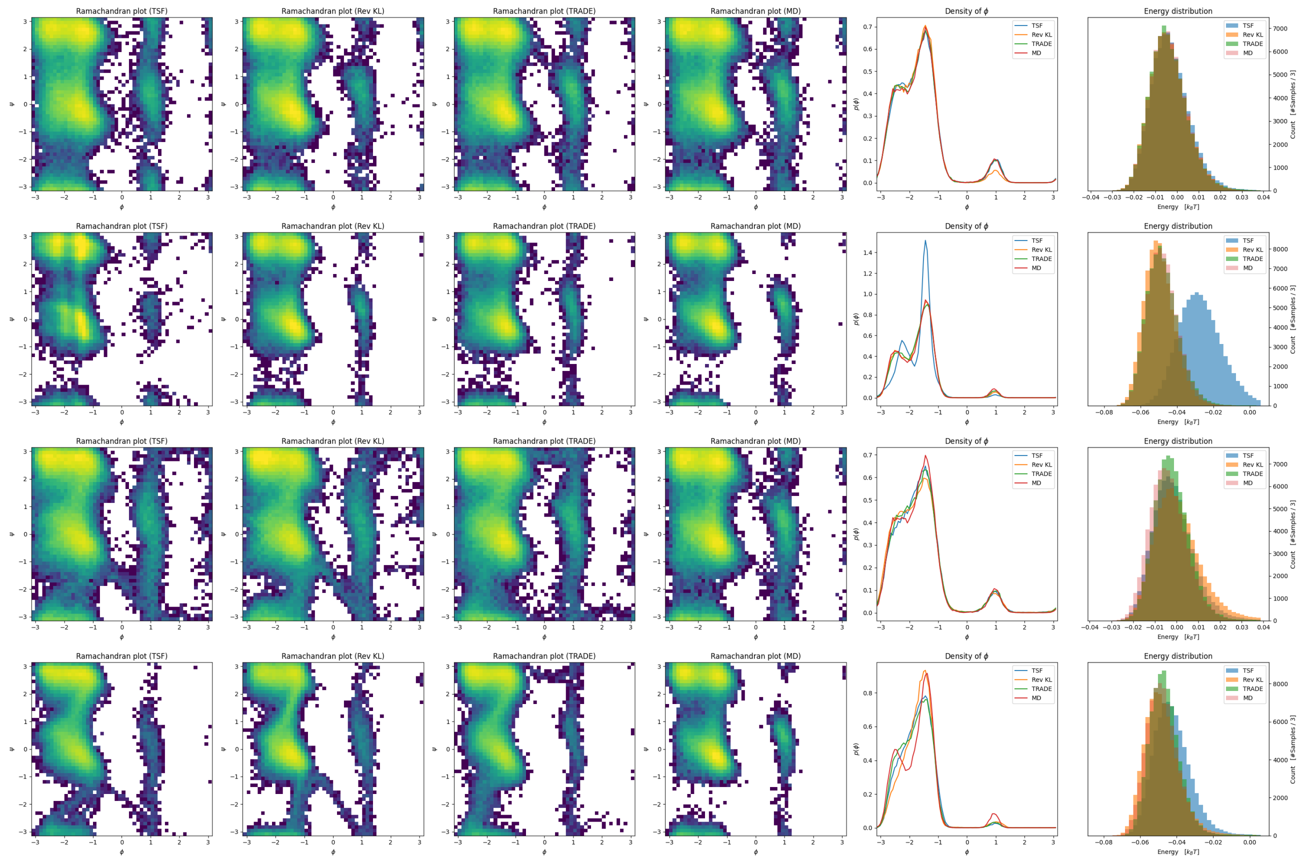

Figure 7: A comparison of temperature steerable flows (TSF, Dibak et al. (2022)) and backward KL with TRADE at different temperatures and architectures for Alanine Dipeptide. From left to right: Ramachandran plots of model samples (TSF, backward KL, TRADE) and molecular dynamics (MD), marginal density of the ϕ angle, and ground truth energy of model samples and MD samples (red). From top to bottom: Spline model $T = 600K$, spline model $T = 300K$, affine model $T = 600K$, affine model $T = 300K$

B.6.2 Model Architecture and Training Details

Model

The three models used in the experiments in Section 5.4 share same model architecture. Instead of directly using the external parameter as condition for the networks, we transform the condition by passing the logarithm of the external parameter through a small residual network. This network consists of one residual block containing a fully connected neural network with two hidden layers, each with 32 hidden neurons.

The invertible function is structured as follows: In the first layer, we apply a channel-wise global linear transformation. The parameters of this transformation are computed by fully connected neural networks with two hidden layers and 64 hidden neurons. The input to these models is the transformed external parameter. After this global scaling, the input images, which have dimensionality $(1, H, W)$, are transformed to a shape of $(4, H/2, W/2)$ using an invertible down-sampling layer (Jacobsen et al., 2018). This layer is followed by three rational-quadratic neural spline coupling blocks (Durkan et al., 2019) which use convolutional subnetworks as described in Table 6.

Next, the data is passed through an invertible flattening operator and three additional rational-quadratic neural spline coupling blocks, this time using fully connected subnetworks (see Table 7). The transformed external parameter is passed to all six rational-quadratic neural spline coupling blocks via an additional channel added to the input of the coupling block. The latent distribution is a 64-dimensional standard normal distribution. All networks use SiLU activations. The weights of the subnetworks used in the coupling blocks are initialized using Xavier normal initialization (Glorot and Bengio, 2010). Additionally, the weights of the final layer in each subnetwork are initialized with zeros.

Table 6: Convolutional subnetworks used in the rational-quadratic neural spline coupling blocks of the normalizing flows for the scalar theory experiments on a lattice, as described in Section 5.4.

Step	Operation	Input shape	Output shape
1	3x3 Convolution	$c_{in} \times H \times W$	$64 \times H \times W$
2	SiLU Activation	$64 \times H \times W$	$64 \times H \times W$
3	3x3 Convolution	$64 \times H \times W$	$64 \times H \times W$
4	SiLU Activation	$64 \times H \times W$	$64 \times H \times W$
5	1x1 Convolution	$64 \times H \times W$	$32 \times H \times W$
6	SiLU Activation	$32 \times H \times W$	$32 \times H \times W$
7	1x1 Convolution	$32 \times H \times W$	$32 \times H \times W$
8	SiLU Activation	$32 \times H \times W$	$32 \times H \times W$
9	1x1 Convolution	$32 \times H \times W$	$c_{out} \times H \times W$

Table 7: Fully connected subnetworks used in the rational-quadratic neural spline coupling blocks of the normalizing flows for the scalar theory experiments on a lattice, as described in Section 5.4.

Step	Operation	Input shape	Output shape
1	Linear	c_{in}	128
2	SiLU Activation	128	128
3	Linear	128	128
4	SiLU Activation	128	128
5	Linear	128	128
6	SiLU Activation	128	128
7	Linear	128	128
8	SiLU Activation	128	128
9	Linear	128	c_{out}

Data

Training data is generated using Langevin dynamic based on the drift term stated in Pawlowski et al. (2017). The MCMC is initialized from Gaussian noise, and we select the states that are in equilibrium. To ensure, that the samples are uncorrelated, we compute the correlation times between the samples and only keep samples that are two correlation times apart from each other.

To obtain training and validation data, we start two different MCMC runs, each with a different random seed.

To ensure that the validation data covers all modes of the data distribution, we use 1,000 samples from the MCMC and apply data augmentation by flipping the sign of the whole states, as well as flipping them horizontally and vertically, along with combinations of these transformations. This results in validation sets of size 8,000 for each validation κ.

For the NLL contribution to TRADE and the combined NLL and backward KL training, we randomly apply data augmentation to the selected batch of training data in each training iteration. This augmentation exploits the symmetries of the system under investigation. Specifically, we randomly flip the sign of the states, apply random mirroring, and random translations of the states.

Training

The hyperparameters for the training with TRADE, NLL-only, and the combined NLL and backward KL objective are summarized in Table 8. The training of TRADE and NLL+backward KL follows a two-stage training: first, the model is only trained using only the NLL loss for n_{nll} epochs. After that, the full objective is applied for the remaining $n_{\text{epochs}} - n_{\text{nll}}$ epochs. In the second phase, the learning rate is multiplied by a decay factor γ_{lr}.

In both phases, a one-cycle learning rate scheduler (Smith and Topin, 2018) is used. Loss balancing, as described in Appendix B.1, is applied to TRADE and the NLL + backward KL training. Additionally, we allow a fixed relative weighting factor λ_{fixed}, which is multiplied by the adaptive λ obtained from the dynamic weighting scheme.

For both training schemes, the range of κ used in the computation of the data-free loss contribution is given by I_κ. For TRADE, this interval is discretized into N_{grid} grid points.

To encourage the model to focus on the region between the two κ values where training data is available, ϵ, as used in the computation of the importance weights (see Section 4.3.2), is reduced in this region for TRADE by multiplying it by γ_ϵ. Across the entire grid, ϵ decays exponentially over the course of training, reaching a final ratio of r_ϵ relative to its initial value. This is intended to broaden the distribution of sampled grid points as the training progresses.

For $p_{\text{grad}}(x|c)$ (see Section 4.3.3) we use $p_\theta(x|\kappa)$. Additionally, we apply the same random data augmentation to the proposed samples as we do to the training data for the NLL contribution and add pixel-wise Gaussian noise with standard deviation σ_{noise}. To prevent training from being affected by outliers, the Huber loss with a parameter δ_{Huber} is used in the computation of $\mathcal{L}_{\text{grad}}$, instead of the mean squared error described in Eq. (19). The exponential averaging of the stored expectation values on the grid points is governed by $\alpha_{\text{expectation}}$ and the exponential averaging of $\mathcal{L}_{\text{grad}}$ at the grid points is governed by α_{grad}.

For the combined NLL and backward KL training, samples from $p_\theta(x|\kappa)$ are required for the backward KL contribution. In the experiment, we sample the logarithm of the condition κ uniformly.

For the training off the NLL-only model the exactly same parameters as for the model trained with TRADE are applied, except that the data-free loss contribution $\mathcal{L}_{\text{grad}}$ is not used.

B.6.3 Model Evaluation

To evaluate the performance of a certain model, we compute the NLL of the validation sets at different values of κ (See entry "Evaluation parameters" in Table 8) and then compute the average of the NLLs across these κ. This average serves as a measure of the model's overall performance. The models are evaluated every five epochs during training, and the model with the lowest average validation NLL is selected for further examinations.

B.6.4 Additional Experimental Results

In this section, we want to examine the stability of the TRADE training scheme. To do this, we use the training configuration described in Table 8 and repeat the training seven times, each with a different random seed. The best model from each run is selected following the procedure outlined in Appendix B.6.3. For every model obtained, we compute the relative ESS as a function of κ. Each ESS value is based on 10,000 samples drawn from the learned distribution. The same evaluation is repeated for the combined NLL and backward KL training.

In Fig. 8, we average the relative ESS over the seven runs for both training schemes (solid lines) and also visualize the interval defined by the standard deviation (shaded regions). For reference, we also plot the ESS for the one model trained with NLL-only on the κ where training data is available.

It is clearly observable, that both training schemes perform best near the κ values where NLL training is performed, with the smallest variation in these regions. However, across the full examined range of κ, TRADE shows significantly smaller fluctuation. In particular, for larger κ values, the combined NLL and backward KL training exhibits a much larger standard deviation.

This evaluation indicates that TRADE produces more reliable training outcomes, showing far less sensitivity to changes in the random seed compared to the baseline of combined NLL and reverse KL training.

Figures 4 and 5 in Section 5.4 are based on the best model (with respect to the average validation NNL) out of the seven runs for each training scheme.

B.7 Software Packages

Our experiments are implemented in Python and rely on the following packages: For the implementation of the training routines, we use PyTorch (Paszke et al., 2019) and PyTorch Lightning (Falcon and The PyTorch Lightning team, 2019). Training progress is tracked by using TensorFlows's (Abadi et al., 2015) TensorBoard. Our invertible neural networks are based on FrEIA (Ardizzone et al., 2022). For general computations, we use NumPy (Harris et al., 2020) and pandas (Wes McKinney, 2010; pandas development team, 2020). Plotting is done with Matplotlib (Hunter, 2007). To systematically test different hyperparameter configurations, we use Optuna (Akiba et al., 2019).

B.8 Compute Resources

To run our experiments, we used up to 12GB of GPU working memory. All our experiments can be completed in less than ten hours on such a device. In addition to our off-the-shelf desktop computers, we had access to a high-performance computing cluster, which we used for some of the experiments.

C GAUSSIAN MIXTURE MODEL

We perform experiments on a Gaussian mixture model to showcase the shortcomings of several methods that were mentioned in the main text. We show the best model obtained with each training method across different temperatures in Fig. 9 and explain the experimental setup and results in the remainder of this section.

C.1 Data Set

The data set used in this experiment is a two-dimensional Gaussian mixture model with six equally weighted modes:

$$p^*((x,y)|c_0) = \frac{1}{6}\sum_{i=1}^{6} \mathcal{N}\left((x,y);\mu_i,\Sigma_i\right). \tag{76}$$

Power-scaling is applied to this distribution, resulting in the following conditional target distribution:

$$p^*((x,y)|c) \propto p^*((x,y)|c_0)^{\frac{c}{c_0}} \tag{77}$$

The covariance matrices Σ_i and the means μ_i of the six Gaussian modes of the target distribution $p^*(x|c_0)$ are chosen as follows:

$$\Sigma_1 = \begin{bmatrix} 0.2778 & 0.4797 \\ 0.4797 & 0.8615 \end{bmatrix} \Sigma_2 = \begin{bmatrix} 0.8958 & -0.0249 \\ -0.0249 & 0.1001 \end{bmatrix} \Sigma_3 = \begin{bmatrix} 1.3074 & 0.9223 \\ 0.7744 & 0.1001 \end{bmatrix}$$

Table 8: Hyperparameter for the models trained on the lattice field theory in Section 5.4. The **first block** corresponds to parameters concerning the general training routine. The **second block** details the specific settings for the data-free loss contribution, and the **third block** describes the dataset and the validation procedure for the trained models. A "-" indicates that this parameter is not applicable the corresponding training scheme.

	TRADE	NLL-only	NLL + backward KL
Learning rate	7.66×10^{-5}	7.66×10^{-5}	1.33×10^{-3}
γ_{lr}	3.0	3.0	0.5
n_{epochs}	400	400	400
n_{nll}	20	20	73
Weight decay	0.0	0.0	0.000135
Batch size (NLL)	1024	1024	1024
Gradient clipping	1,000	1,000	16.0
Optimizer	Adam	Adam	Adam
λ_{fixed}	0.355	-	3.847
σ_{noise}	0.38	-	-
I_κ	$[0.22, 0.32]$	-	$[0.22, 0.32]$
δ_{Huber}	6.87	-	-
Batch size (data-free loss)	2048	-	2048
ϵ causality weights	1.57×10^{-6}	-	-
γ_ϵ	0.376	-	-
r_ϵ	0.228	-	-
N_{grid}	150	-	-
$\alpha_{\text{expectation}}$	0.1	-	-
α_{grad}	$0\ 2.5 \times 10^{-5}$	-	-
α_{balance}	0.1	-	0.1
Update frequency expectation value	2,000 iterations	-	-
Samples expectation value update	1500	-	-
Base parameters for NLL loss	$\{0.25, 0.29\}$	$\{0.25, 0.29\}$	$\{0.25, 0.29\}$
Independent samples per base parameter	100,000	100,000	100,000
Samples per validation parameter	8,000	8,000	8,000
Evaluation parameters	$\{0.24, 0.25, ..., 0.3\}$	$\{0.24, 0.25, ..., 0.3\}$	$\{0.24, 0.25, ..., 0.3\}$
Random data augmentation (training)	yes	yes	yes
Dequantization noise	no	no	no
Lattice size	8×8	8×8	8×8

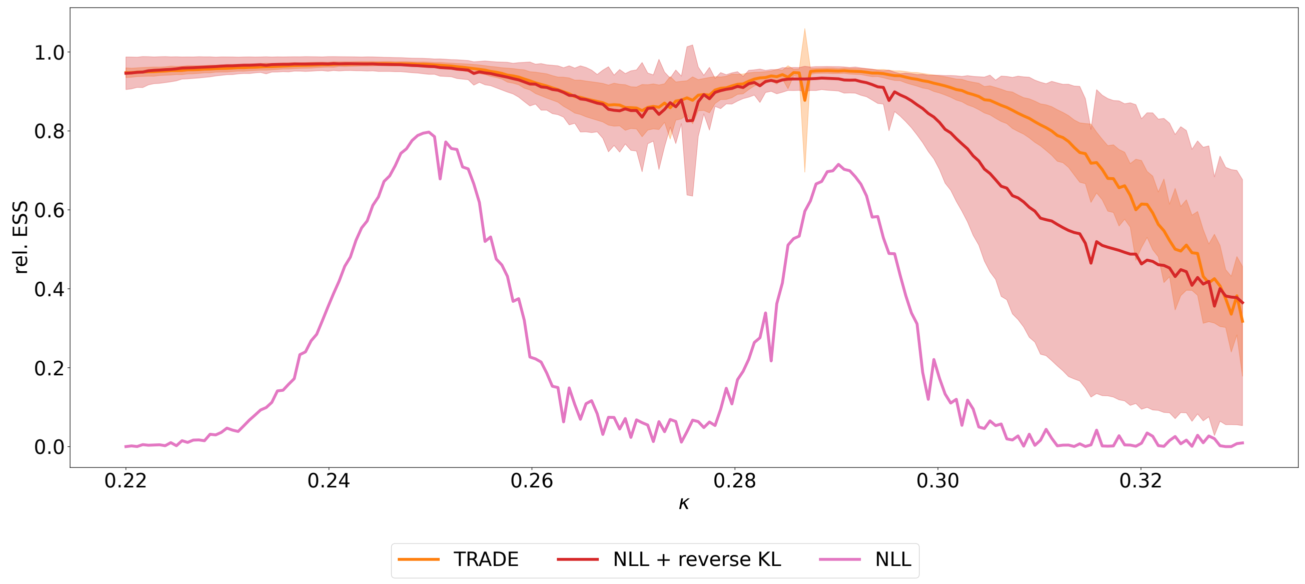

Figure 8: Relative ESS for the different training schemes examined for the scalar theory on an 8×8 lattice. For the model trained with TRADE and with a combination of NLL and backward KL, the solid line indicate the average over seven training runs, each initialized with a different random seed. The shaded areas correspond to the interval defined by the standard deviation of the ESS. For reference, the single run with NLL-only training is included as well. On can clearly observe that TRADE is much less sensitive to the selection of the random seed compared to the combined NLL and backward KL training.

$$\Sigma_4 = \begin{bmatrix} 0.0305 & 0.0142 \\ 0.0142 & 0.4409 \end{bmatrix} \Sigma_5 = \begin{bmatrix} 0.0463 & 0.0294 \\ 0.0294 & 0.3441 \end{bmatrix} \Sigma_6 = \begin{bmatrix} 0.1500 & 0.0294 \\ 0.0294 & 1.5000 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} -1.0 \\ 2.0 \end{bmatrix} \mu_2 = \begin{bmatrix} 3.0 \\ 7.0 \end{bmatrix} \mu_3 = \begin{bmatrix} -4.0 \\ 2.0 \end{bmatrix} \mu_4 = \begin{bmatrix} -2.0 \\ -4.0 \end{bmatrix} \mu_5 = \begin{bmatrix} 0.0 \\ 4.0 \end{bmatrix} \mu_6 = \begin{bmatrix} 5.0 \\ -2.0 \end{bmatrix}$$

C.2 Models

C.2.1 Non-Volume-Preserving Flow

The non-volume-preserving flows are implemented as four consecutive Rational Quadratic Spline coupling blocks (Durkan et al., 2019). The logarithm of the external parameter c is incorporated into each coupling block by concatenating it with the block's input. The hyperparameters of the Rational Quadratic Spline transformations are set to the default values of FrEIA's (Ardizzone et al., 2022) implementation of the Rational Quadratic Spline coupling block. For the model trained with the negative log-likelihood method at c_0 only, the invertible function is not conditioned on the external parameter c. Therefore, in this model, no conditioning information is passed to the coupling blocks. Each coupling block is preceded by an ActNorm layer and followed by a fixed random permutation of the output dimensions. This ensures that the active and passive dimensions are alternated in each coupling block. The subnetworks used to predict the transformation parameters in each coupling block are fully connected neural networks with two hidden layers of width 32 and SiLU activation. The weights of the linear transformations in these networks are initialized using PyTorch's Xavier normal initialization scheme (Glorot and Bengio, 2010). The weights and biases of the output layer in each subnetwork are initialized to zero to ensure that the invertible function resembles the identity function at the start of the training. The latent distribution is a power-scaled standard normal distribution:

$$p_0(z|c) \propto \mathcal{N}\left(z; 0, \mathbf{1}_2\right)^{\frac{c}{c_0}} . \tag{78}$$

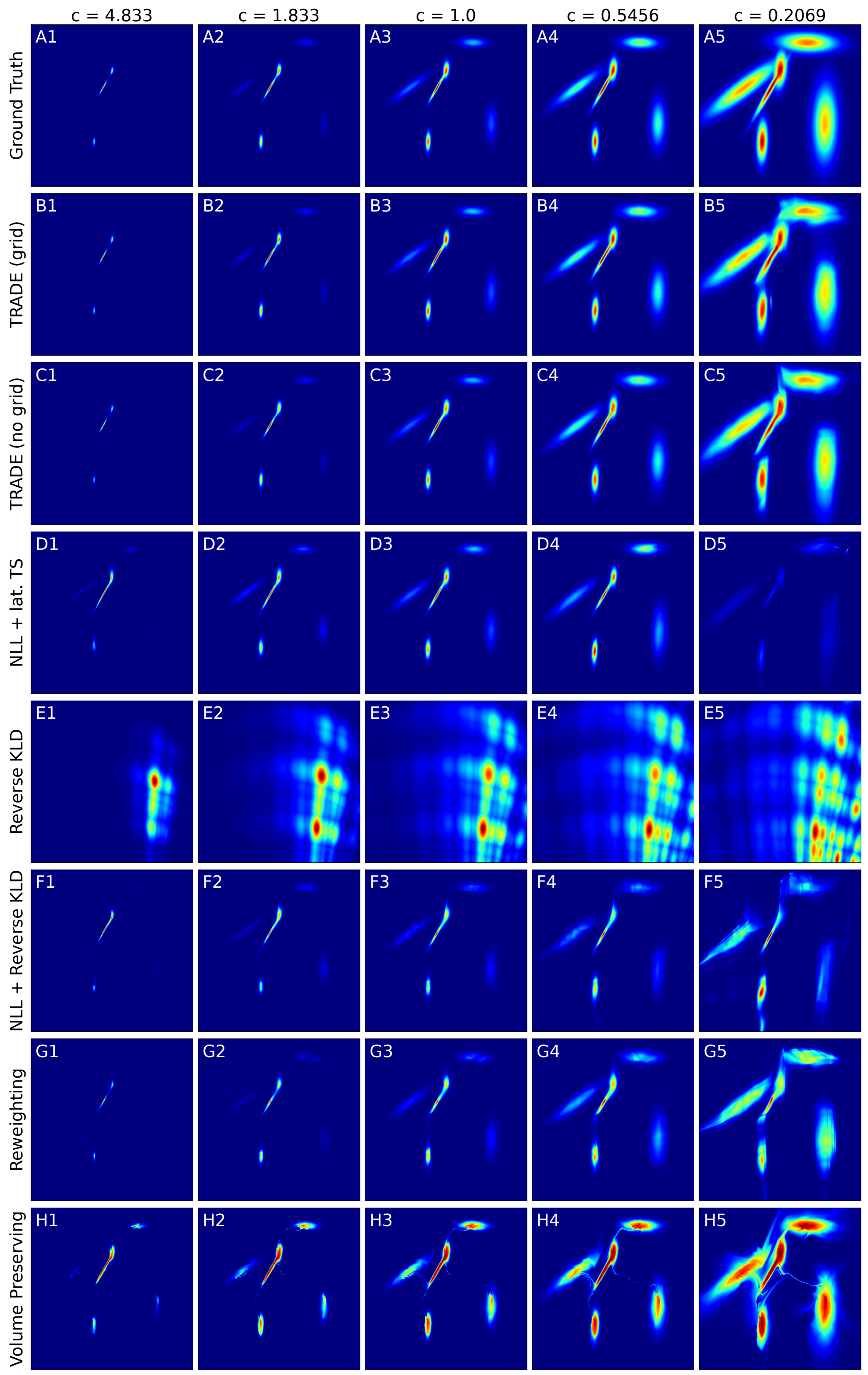

Figure 9: Visualization of the best learned densities for coupling based normalizing flows trained with different training objectives in comparison to the ground-truth density (**A**) for the best performing model based on the average validation negative log-likelihood (nll) at different external parameters. **B**: Model trained using TRADE with a discretized parameter space. **C**: TRADE trained with a continuous parameter space. **D**: Only nll training on training data at $c_0 = 1.0$ with latent power-scaling **E**: Reverse KL training **F**: Combined nll and reverse KL training. **G**: Training using reweighting of training samples at $c_0 = 1.0$ **H**: Volume-Preserving normalizing flow trained with nll training at c_0 and latent power-scaling.

C.2.2 Volume-Preserving Flow

The volume-preserving flow consists of 20 GIN coupling blocks (Sorrenson et al., 2020) and uses a learnable permutation after each coupling block. The subnetworks and their initialization is the same as those used for the Rational Quadratic Splines, except that ReLU activation is used instead of SiLU activation. No condition is ingested to the coupling blocks. The latent distribution is a power-scaled standard normal distribution.

C.3 Training

All models are trained using the Adam optimizer (Kingma and Ba, 2017) with a maximum learning rate of $1 \cdot 10^{-4}$, modulated by a oneCycle learning rate scheduler (Smith and Topin, 2018). A total of $100,000$ training samples following $p(x|c_0 = 1.0)$ are used to compute the negative log-likelihood loss for all training objectives except pure reverse KL training. The batch size for the negative log-likelihood loss contribution is 256 and the batch size for all data-free losses is set to 512. Training runs for a total of 600 epochs. Whenever the negative log-likelihood loss is combined with a data-free loss contribution (i.e., TRADE and reverse KL combined with negative log-likelihood training) the adaptive loss balancing scheme described in Appendix B.1 is applied. The estimates $\hat{\lambda}_{\text{boundary}}$ and $\hat{\lambda}_{\text{grad}}$ are smoothed with an exponential averaging scheme, with smoothing parameter $\alpha_{\text{balance}} = 0.015$. For all models, gradient clipping with a threshold of 2.0 is applied. For all data-free loss contributions (i.e. TRADE and reverse KL), the parameters at which the loss is computed are sampled from the interval

$$[c_{\min}, c_{\min}] = [0.16238, 6.15848] \tag{79}$$

.

If the negative log-likelihood loss is combined with a data-free loss, both losses are jointly optimized from the beginning of the training. Besides these general training settings, the following loss-specific hyperparameters are applied:

C.3.1 TRADE With Grid

The condition is discretized into a grid of 250 points, initialized uniformly in the logarithmic space. The parameter governing the decay rate in the adaptive distribution for the condition values, at which the physics-informed loss is evaluated, is set to $\epsilon = 1.0$ and is decayed to $\epsilon = 0.8$ over the course of training using an exponential decay scheme. The losses stored for each grid point are smoothed using an exponential averaging scheme with a smoothing parameter $\alpha_{\text{loss}} = 0.9$. The stored expectation values are updated every 1,000 training steps based on 1,000 samples. These expectation values are also smoothed using an exponential averaging scheme with a parameter $\alpha_{\text{expectation}} = 0.75$. Instead of the means squared loss, the physics-informed loss contribution utilizes the Huber loss with a parameter $\delta = 0.1$. In each training step, the physics-informed loss contribution is evaluated at 512 condition values c sampled from the adaptive distribution p_{grad}. For each condition value, one evaluation point is drawn from $p_\theta(x|c)$.

C.3.2 TRADE Without Grid

The condition c is sampled uniformly in the logarithmic domain, with the borders of the interval from which the parameters are sampled linearly increasing in the logarithmic domain over time, reaching the full range by the end of the training, after 600 epochs. In each training step, five different parameters c are sampled, at which the physics-informed loss contribution is evaluated for 105 evaluation points following $p_\theta(x|c)$. For the approximation of the expectation values required in the physics-informed loss contribution, 500 samples are used for each value of c. The remaining parameters follow those used in the grid-based version of TRADE.

C.3.3 Reverse KL Training

For the computation of the reverse KL objective, 512 condition values c are sampled uniformly in the logarithmic space. These samples are then used to approximate the reverse KL objective. Unlike for TRADE, where the sampling range expands gradually, here the full range of possible condition values is considered from the beginning of the training.

C.4 Model Selection

To evaluate the performance of the model at certain stages of training, the average validation negative log-likelihood is monitored and computed every five epochs. Based on this criterion, the best-performing model is selected. For this evaluation, 10,000 data points are generated at 15 different evaluation parameters. The evaluation parameters are equally spaced in the logarithmic domain in the interval $[0.2069, 4.8330]$. Since sampling from the power-scaled target distribution is not straight forward, rejection sampling is applied. To ensure that the proposal distribution has sufficient overlap with the target distribution-and that enough proposal samples are obtained at the tails of the target distribution for small values of c-Gaussian noise with known variance and mean is added to the proposal states drawn from the target distribution $p(x|c_0)$ (sampling from $p(x|c_0)$ is simple as it is a Gaussian mixture model). This approach corresponds to convolving each mode of $p(x|c_0)$ with a normal distribution, which results in another normal mode. This allows for an easy evaluation of the density of the convolved distribution, as required for rejection sampling.

C.5 Results

For the best model from each training run, the learned densities are visualized in Fig. 9. Examining the two models trained with the TRADE objective (subplots B1 to B5 for the grid-based version of TRADE and C1 to C5 for the grid-less version of TRADE), it is evident, that the models have successfully learned the distribution from the available training data at $c_0 = 1.0$ (see subplots B3 and C3). A visual comparison between the target and the learned distributions suggests that the models have also captured how the distribution scales with the external parameter. The number of learned modes matches the target across all visualized parameter values, and the shape and relative brightness (and therefore height) are consistent with the target. However, for the smallest condition values (subplots B5 and C5), a slight mismatch in the shape of the learned modes can be observed. For the model trained only with negative log-likelihood training at $c_0 = 1.0$ (subplots D1 to D5), with latent power-scaling, it can be observed that while the number and position of modes are learned correctly, their relative height deviates from the target. This is expected, as the flow is non-volume-preserving, and there is no reason to assume that simply scaling the latent distribution would result in the correct scaling in data space. The model trained with reverse KL training (subplots E1 to E5) completely fails to reproduce the characteristics of the target distribution: The number of modes is incorrect, and both their positions and their shapes are misrepresented. While the widening of the distribution as c increases is evident, the relative weighting of the individual modes is inconsistent over different values of c. For instance, in subplots E1 and E4, the highest modes appear to swap positions. The model trained with combined reverse KL and negative log-likelihood objective (subplots F1 to F5) captures the general characteristics and scaling properties of the target. However, for smaller values of c, the relative heights and shapes of the learned modes deviate from the target. Examining the model trained with reweighting, it can be observed that for $c > 1.0$, where the target distribution is narrower than the distribution at c_0 from which the training data was sampled, the model closely matches the target. However, for smaller values of c (e.g. subplot G5), the distribution at c_0 is narrower than the target at c, providing only little training signal in the region of the tails of the target distribution. Consequently, the learned distribution's tails are too narrow in this regime of condition values compared to the target distribution. Looking at the remaining model, the volume-preserving flow with latent power-scaling, it is evident, that as expected, the scaling with c is correctly modeled based on the distribution learned at c_0. However, the relative heights of the different modes are inaccurate (see, for example, subplot H1) compared to the target. Additionally, the learned modes are connected by high-density bridges (see, for example, subplot H5).

This qualitative evaluation of the performance of the different training approaches is supported by analyzing the average validation (forward) KL divergence between the target distribution and the learned distribution, as presented in Table 9. The values in this table are based on 80,000 validation samples drawn from the target distribution at each of the specified condition values. The uncertainties shown in the table are approximated using bootstrapping, with 20 resampled sets.

After evaluating the *best* performing mode for each training run, it is also insightful to examine the *final* model of each training run. The visualization of the densities for these snapshots of the training are presented in Fig. 10. For the two models trained with TRADE, the model trained with reweighting, the model trained with negative log-likelihood training only, and the volume-preserving flow, the densities exhibit only minor differences between

	KLD $c = 4.8330 \downarrow$	KLD $c = 2.9764 \downarrow$	KLD $c = 1.8330 \downarrow$
TRADE (grid)	**0.011482±0.000487**	0.009177±0.000393	0.006239±0.000407
TRADE (no grid)	0.011761±0.00054	**0.00746±0.000422**	**0.005112±0.000301**
NLL + lat. TS	0.58906±0.0023	0.37628±0.00279	0.14435±0.00176
Reverse KLD	9.19012±0.00337	6.45853±0.00596	4.63207±0.00463
NLL + Reverse KLD	0.32929±0.00299	0.229±0.0023	0.11099±0.00169
Reweighting	0.11298±0.0014	0.09674±0.00122	0.09815±0.00127
Volume Preserving	0.80099±0.00236	0.55106±0.00293	0.26806±0.00274
	KLD $c = 1.1288 \downarrow$	KLD $c = 1.0 \downarrow$	KLD $c = 0.8859 \downarrow$
TRADE (grid)	0.003704±0.000284	0.004421±0.00029	0.004042±0.000397
TRADE (no grid)	**0.002825±0.000299**	**0.003702±0.000323**	**0.003906±0.000357**
NLL + lat. TS	0.022951±0.000657	0.019051±0.000589	0.032336±0.00087
Reverse KLD	3.56005±0.00668	3.37188±0.00591	3.08986±0.00583
NLL + Reverse KLD	0.04906±0.000899	0.049896±0.000861	0.06853±0.00115
Reweighting	0.07873±0.0011	0.07396±0.00151	0.074053±0.000894
Volume Preserving	0.11362±0.00201	0.09766±0.00153	0.05473±0.00172
	KLD $c = 0.5456 \downarrow$	KLD $c = 0.3360 \downarrow$	KLD $c = 0.2069 \downarrow$
TRADE (grid)	0.013757±0.000584	**0.035±0.00133**	**0.08647±0.0025**
TRADE (no grid)	**0.013047±0.000637**	0.03706±0.00159	0.08707±0.00146
NLL + lat. TS	0.16599±0.00211	0.46467±0.0031	0.92979±0.00628
Reverse KLD	2.90968±0.00406	2.78683±0.00498	2.75597±0.00506
NLL + Reverse KLD	0.14625±0.00254	0.34991±0.00488	0.74761±0.00944
Reweighting	0.06681±0.00152	0.16942±0.00286	0.52953±0.00739
Volume Preserving	0.09508±0.00208	0.15019±0.00351	0.2335±0.00415

Table 9: Validation KL divergence between the target distribution and the models trained with various objective functions evaluated at different external parameters. The models selected for this evaluation are the best performing models under the average validation negative log-likelihood. Bold numbers indicate the best performing model at each condition value. The rows highlighted in gray represent the entries for the models trained with the TRADE approach. In this experiment TRADE outperforms the evaluated base line methods at each evaluated external parameter.

Fig. 9 and Fig. 10. However, for the two models trained using the reverse KL objective, a significant change can be observed: In Fig. 9, the model trained with reverse KL alone shows no resemblance to the target density. In contrast, in Fig. 10, it has learned some of the modes correctly—one mode for the three largest values of c, and four or five modes for the two smallest values of c, respectively. For the model trained with the combination of reverse KL and negative log-likelihood training, all modes are captured in Fig. 9 but are subsequently forgotten in Fig. 10. Similar to the model trained with reverse KL alone, this issue is particularly sever for large values of c. The only exception is $c = 1.0$, where negative log-likelihood training was performed, ensuring that all modes are properly learned. This observation confirms the well-known issue of mode-seeking behavior in reverse KL training. Notably, for the combined reverse KL and nll training, this result suggests that providing training data does not have a stabilizing effect on the model but merely prevents mode collapse at the specific parameter value where training data is available.

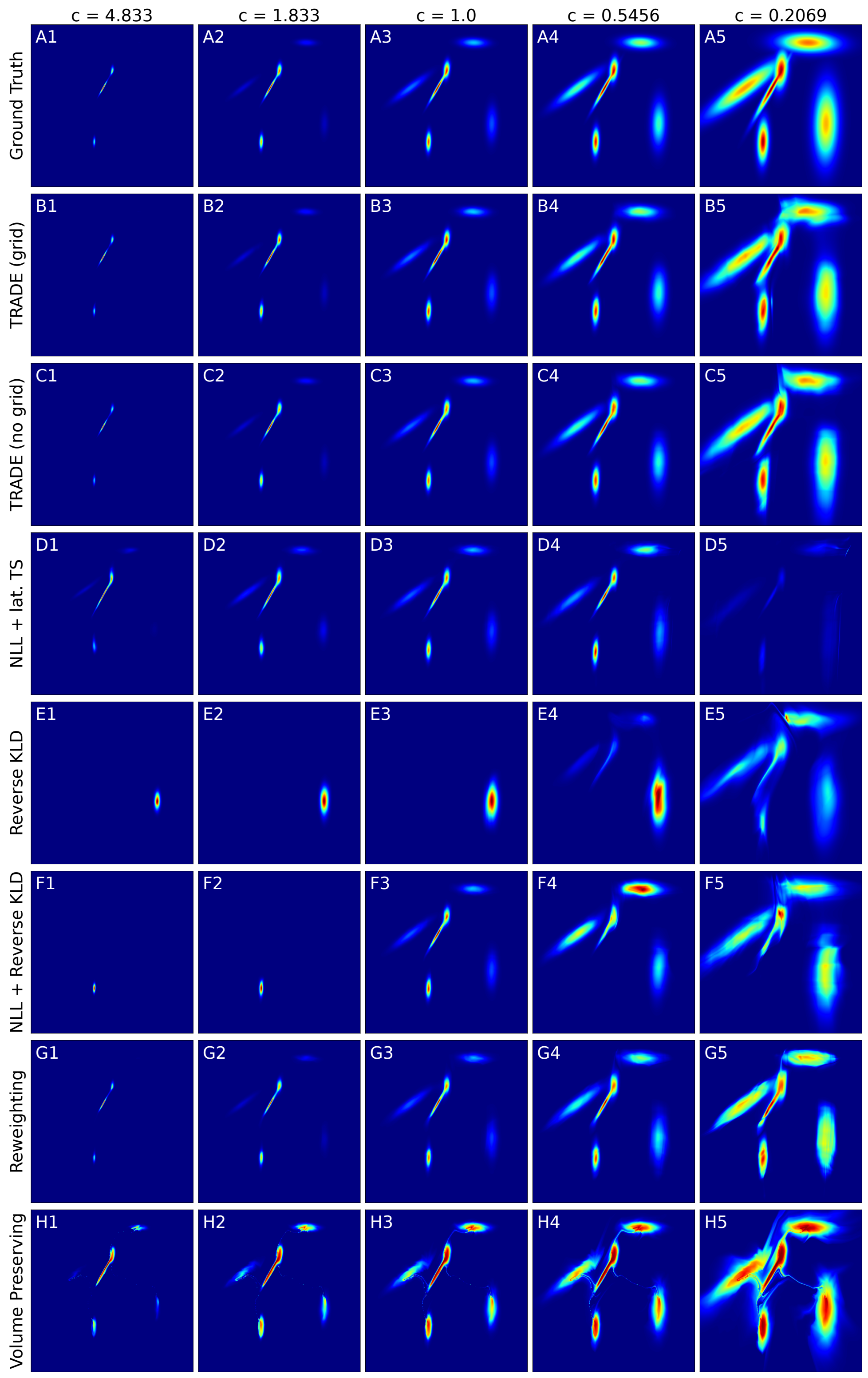

Figure 10: Visualization of the final learned densities for coupling based normalizing flows trained with different training objectives in comparison to the ground truth density (**A**) for model observed at the end of the training run evaluated at different external parameters. **B**: Model trained using TRADE with a discretized parameter space. **C**: TRADE trained with a continuous parameter space. **D**: Only nll training on training data at $c_0 = 1.0$ with latent power-scaling **E**: Reverse KL training **F**: Combined nll and reverse KL training. **G**: Training using reweighting of training samples at $c_0 = 1.0$ **H**: Volume-Preserving normalizing flow trained with nll training at c_0 and latent power-scaling.