Department of Physics and Astronomy

Ruprecht Karl University of Heidelberg

Master Thesis in Physics

submitted by

Lars Erik Kühmichel

born in Chemnitz, Germany

2024

Advancements in Context-Aware Learning and Generative Modeling

This thesis was carried out by

Lars Erik Kühmichel

at the

Interdisciplinary Center for Scientific Computing

under the supervision of

apl. Prof. Dr. Ullrich Köthe

Zusammenfassung

In dieser Arbeit erforschen wir hochmoderne generative Modelle, indem wir ihre entsprechenden Trainingstechniken analysieren und verbessern. Im Zuge dessen leisten wir einen empirischen Beitrag zu [1]. Hier analysieren wir die Konvergenzrate von GAUSSIANIZATION, einer Variante des NORMALIZING FLOW, sowohl auf Beispiel- als auch realen Datensätzen. Weiterhin formulieren wir einen neuen Ansatz zur Ausrichtung von Flusstrajektorien in FLOW MATCHING, mittels mini-batch optimalem Transport. Schließlich wenden wir permutationsinvariante neuronale Netze an, um kontextbewusste Modelle sowohl in einem generativen als auch in einem klassisch überwachten Kontext für die Domänengeneralisierung zu trainieren. Dadurch tragen wir maßgeblich zu [2] bei.

Abstract

In this thesis, we explore cutting-edge generative models by analysing and refining their corresponding training techniques. In doing so, we contribute to [1] empirically by examining the convergence rate of GAUSSIANIZATION, a variant of NORMALIZING FLOW, across toy and real-world datasets. Additionally, we devise a novel approach to straightening flow trajectories in FLOW MATCHING, using mini-batch Optimal Transport. Lastly, we employ permutation-invariant neural networks to train context-aware models in both generative and classically supervised contexts for Domain Generalization. This presents a significant contribution to [2].

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Declaration:

I hereby certify that I have written this thesis independently and did not use any sources or tools other than those indicated.

Heidelberg, den 24.01.2024 ....

Lars Erik Kühmichel

Contents

1 Introduction

In recent years, Generative Modeling has proven itself to be an effective tool for the analysis and processing of complex, high-dimensional data. However, while model architectures and training techniques continually improve, real-world data corruption and perturbations often still present a failure case for deep learning techniques [3], [4].

In this work, we explore state-of-the-art generative models, analysing and improving their training techniques. We contribute to [1] empirically by exploring the convergence rate of GAUSSIANIZATION on high-dimensional, real-world datasets.

Furthermore, we contribute to [2], employing permutation-invariant neural networks as set-encoders [5], [6] to give our models contextual information, improving their generative or predictive capabilities under distribution shift.

Specifically, we consider a class of "context-aware" models, where a prediction is made given a contextual embedding c, which is inferred from a set of "similar" samples $\mathcal{S}^{(n)}$:

$$p\left(x; c = E\left(\mathcal{S}^{(n)}\right)\right) \tag{1}$$

$$p\left(y; x, c = E\left(\mathcal{S}^{(n)}\right)\right) \tag{2}$$

for a set-encoder E in the generative and classically supervised settings, respectively.

Particularly in the framework of Domain Generalization [7], [8], where data is available from distinct environments[1], we can collect $\mathcal{S}^{(n)}$ as a set of samples from the same domain $\mathcal{D}_0$ as the singleton sample x_0:

$$\mathcal{S}^{(n)} = \{x_i \mid \mathcal{D}_i = \mathcal{D}_0\}_{i=1}^{n} \tag{3}$$

In the generative setting, we employ FLOW MATCHING [9], [10] to generate 3-dimensional point clouds, supplying a context embedding encoded from the shape of a whole point cloud [11], [12]. We formulate a new approach to straighten the flow trajectories within FLOW MATCHING, improving sampling speed and reducing training objective stochasticity while retaining sample quality over the method proposed in the original paper [9].

Within [2], we focus on Domain Generalization, formalising three criteria for when the set-based contextual embedding is beneficial. We verify our criteria empirically with

[1]We use the terms environment and domain interchangeably.

classically supervised models, aiming to detect potential failure cases. This makes our models robust to distribution shifts by allowing us to apply a selection between a highly adapted in-distribution model and a highly robust domain-invariant model.

In summary, the contributions highlighted in this thesis are

1. We contribute empirically to [1], determining the scaling behaviour of the convergence rate of GAUSSIANIZATION for toy and real-world datasets.
2. We propose a novel approach to straightening flow trajectories in FLOW MATCHING via mini-batch Optimal Transport assignment.
3. We employ permutation-invariant neural networks to train a high-fidelity, context-aware generative model for 3-dimensional point clouds.
4. We contribute to [2], performing a novel approach to Domain Generalization by selecting the optimal model based on failure case detection.

1.1 Abbreviations

Abbreviation	**Description**
AUROC	Area under Receiver Operating Characteristic Curve
CAFM	Context-Aware Flow Matching
CNF	Continuous Normalizing Flow
CNN	Convolutional Neural Network
DG	Domain Generalization
FM	Flow Matching / Rectified Flow
ID	In-distribution
IID	Independent and Identically Distributed
MMD	Maximum Mean Discrepancy
NF	Normalizing Flow
ODE	Ordinary Differential Equation
OOD	Out-of-distribution
PCA	Principal Component Analysis
SOTA	State-of-the-art

1.2 Mathematical Notation

Notation	Description
x	A data sample
z	A latent sample
$p(x)$	The probability of x
$p(x; c)$	The probability of x conditioned on c
Q	A rotation matrix $Q \in \mathrm{SO}(n)$
$p_t(x)$	The probability of x at time t, equivalent to $p(x; t)$
x_0	The singleton sample
$\mathcal{S}^{(n)}$	A set of n samples related to the singleton sample x_0
$\mathcal{D}_0$	The domain label of the singleton sample x_0
$\mathcal{D}_i$	The domain label of any other arbitrary sample x_i

2 Background

This section aims to establish a knowledge baseline essential to understanding the development of methods and new ideas in Section 3, as well as the setup of experiments in Section 4. Readers with a strong background in deep learning, or those already familiar with the current state-of-the-art may wish to skip ahead to Section 3.

2.1 Generative Modeling

Generative Modeling is a fundamental, and currently highly active, branch of research in deep learning. It centres around training models to understand complex, and often high-dimensional, probability distributions. This is typically achieved by learning an invertible mapping F that transforms samples between this complex data distribution $p(x)$ and a simple latent prior $p(z)$, often chosen as the standard normal $\mathcal{N}(0, \mathbb{I})$. This enables sampling from $p(x)$ (i.e., generating new data) by simply applying the inverse mapping:

$$x = F^{-1}(z) \tag{4}$$

The central idea behind Generative Modeling goes much further than simply sampling, however. Observe that the trained models contain vastly fewer parameters than the total number of parameters in the dataset. As such, instead of being able to simply memorise every data point, generative models are forced to recognise relationships and patterns between features. This allows us to leverage generative models for applications like denoising, inpainting, structured prediction, and many more [13].

2.2 Domain Generalization

Domain Generalization [7], [8] concerns itself with a data setting where samples come from many distinct environments with differing characteristics. Consider, for instance, a dataset that contains images of dogs. Such images can come in many forms, such as photographs, sketches, oil paintings, computer-generated renders, etc.

Notably, the visible features of these images differ vastly, but humans are inherently able to generalise between these distributions and recognise such images as depictions of dogs.

However, for deep learning models, unseen distribution shifts often still present a significant failure case [3], [4]. Since it can be difficult, or even impossible, to incorporate every possible distribution shift into training, Domain Generalization aims to build models that are robust to such unseen shifts.

2.3 Optimal Transport

While Generative Modeling concerns itself with *how* to transport samples from one distribution to another, Optimal Transport takes this notion one step further, asking how to transport these samples *while minimising some transport cost* c.

To illustrate this, envision a set of identical particles floating around in a room. The room is initially brightly lit; the position of all particles is clearly visible. We turn the light off for a short moment, and when it comes back on, the positions of the particles have shifted.

Finding out which particle is which is impossible since they are identical. However, we can make an educated guess based on how the particles may have moved. Unfortunately, as illustrated in Figure 1, there are an infinite number of possible solutions for the movement of the particles.

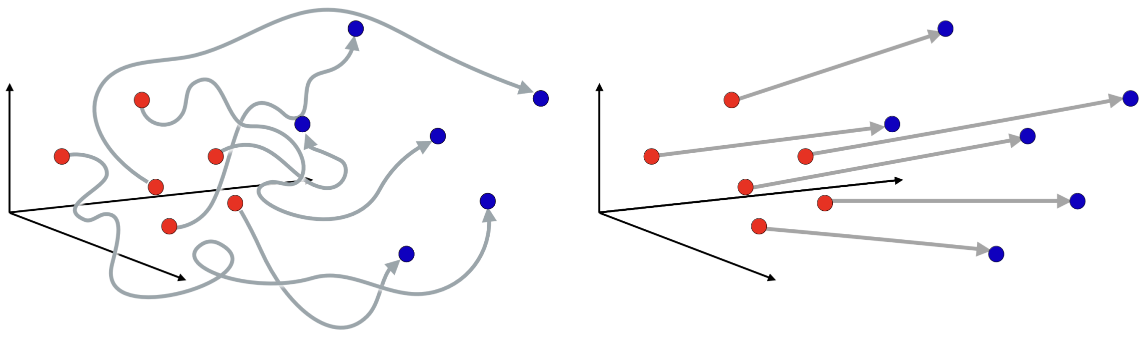


Figure 1: *(Left)* The solution to the transport problem from the red distribution to the blue distribution is not unique. *(Right)* The Optimal Transport solution is unique under Euclidean cost. Images taken from [14].

We can reasonably simplify the question by instead asking how the particles may have moved, assuming the *principle of least action*. Under the Euclidean transport cost, the solution is then unique, allowing only the shortest total distance on straight line paths.

There are a host of algorithms that solve Optimal Transport under various conditions, and correspondingly, theory to go alongside them. For brevity, we will only consider Optimal Transport under Euclidean transport cost using the Sinkhorn-Knopp algorithm as presented in Section 3.

2.4 Normalizing Flows

Normalizing Flows are a class of generative models trained with the maximum likelihood principle. They are typically implemented as a composition of n simple, analytically invertible transforms f_i:

$$F(x) = f_n \circ \ldots \circ f_1(x) \tag{5}$$

The likelihood can then be computed exactly via the change of variables formula:

$$\log p(x) = \log p(z) + \log \det J_F \tag{6}$$

where $\det J_F$ is the Jacobian determinant of the flow F. Since we chose F as a composition, we now only require each f_i to have a tractable Jacobian determinant:

$$\log \det J_F = \sum_i \log \det J_{f_i} \tag{7}$$

The variants of Normalizing Flows we cover here present several advantages. Firstly, Normalizing Flows are highly efficient, allowing inference within a single network forward pass. They offer tractable and exact likelihood computation and, when trained end-to-end, further possess strong disentanglement capabilities, much in contrast to diffusion models.

However, these merits come at the cost of expressiveness, since the Jacobian is constrained due to the application of restricted network architectures. As such, Normalizing Flows often struggle to learn high-fidelity representations of complex, high-dimensional data like photographs. The resulting distributions may also exhibit artefacts, such as bridges between modes of the likelihood.

Recent work on Normalizing Flows has focused on removing the architectural restrictions [15], [16], promising highly expressive free-form transforms within a single inference step.

2.4.1 Coupling Flows

Coupling Flows, as introduced in [17], [18], are currently the perhaps most widely used form of Normalizing Flows. In a Coupling Flow, each transform f is coupled, meaning that it conditions on one part of the data while only transforming the other:

$$f(x_1, x_2) = \operatorname{concat}\big(x_1, \tilde{f}(x_2; x_1)\big) \tag{8}$$

A common yet simple choice for $\tilde{f}$ is the affine transform:

$$\tilde{f}(x_2; x_1) = s(x_1)x_2 + t(x_1) \tag{9}$$

where both s and t can be neural networks.

The primary advantage of this formulation is that the Jacobian determinant required for (6) becomes trivial to compute in a single network forward pass:

$$\begin{aligned}
\det J_f &= \det \begin{pmatrix} \frac{\partial x_1}{\partial x_1} & \frac{\partial x_1}{\partial x_2} \\ \frac{\partial \tilde{f}(x_2;x_1)}{\partial x_1} & \frac{\partial \tilde{f}(x_2;x_1)}{\partial x_2} \end{pmatrix} \\
&= \det \begin{pmatrix} I & 0 \\ \frac{\partial \tilde{f}(x_2;x_1)}{\partial x_1} & \frac{\partial \tilde{f}(x_2;x_1)}{\partial x_2} \end{pmatrix} \\
&= \det \frac{\partial \tilde{f}(x_2; x_1)}{\partial x_2} = \det s(x_1)
\end{aligned} \tag{10}$$

Notably, $\tilde{f}$ is analytically invertible since s and t are only ever evaluated in the forward direction:

$$\tilde{f}^{-1}(y_2; x_1 = y_1) = \frac{y_2 - t(x_1)}{s(x_1)} \tag{11}$$

A visual representation can be found in Figure 2.

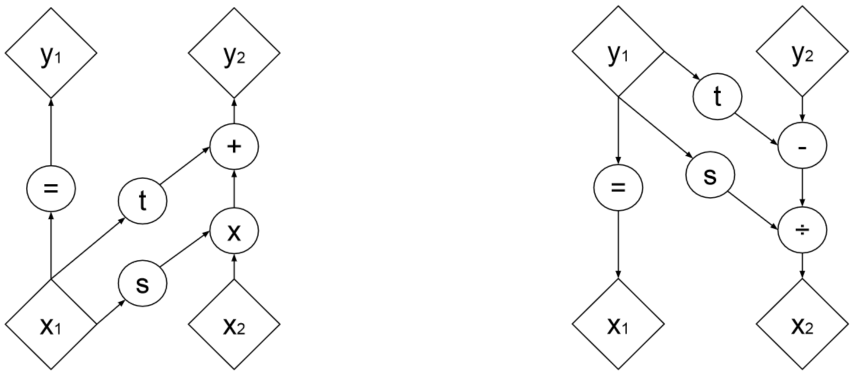


Figure 2: Visualisation of (9). *(Left)* Forward pass. *(Right)* Inverse pass. s and t are only ever evaluated in the forward direction and may thus be neural networks. Image taken from [18].

To ensure that all dimensions are treated an (approximately) equal number of times by the flow, x_1 and x_2 are typically swapped, randomly permuted, or rotated after each layer. More recent research suggests that using strongly expressive couplings, like rational-quadratic splines [19] can help mitigate some of the shortcomings of the restricted architecture.

2.4.2 Gaussianization

Gaussianization, originally proposed by [20], is a variant of Normalizing Flows where the data is transformed via a series of 1-dimensional transforms. Contrary to Coupling Flows, each transform is here entirely unconditioned, such that f becomes an elementwise transform. As such, the affine transform, as shown in (9), would become:

$$f(x) = sx + t \tag{12}$$

Where now s and t are network parameters rather than networks themselves. Similar to coupling flows, Gaussianization ensures an equal treatment of all dimensions by rotating the data after each step. [21] originally proposed a host of rotation methods. In this work, we solely consider random rotations $Q \sim \mathrm{SO}(n)$. Modern applications for Gaussianization, like [22] or [23], use data-dependent rotations such as PCA, max K-SWD, or learned rotations instead.

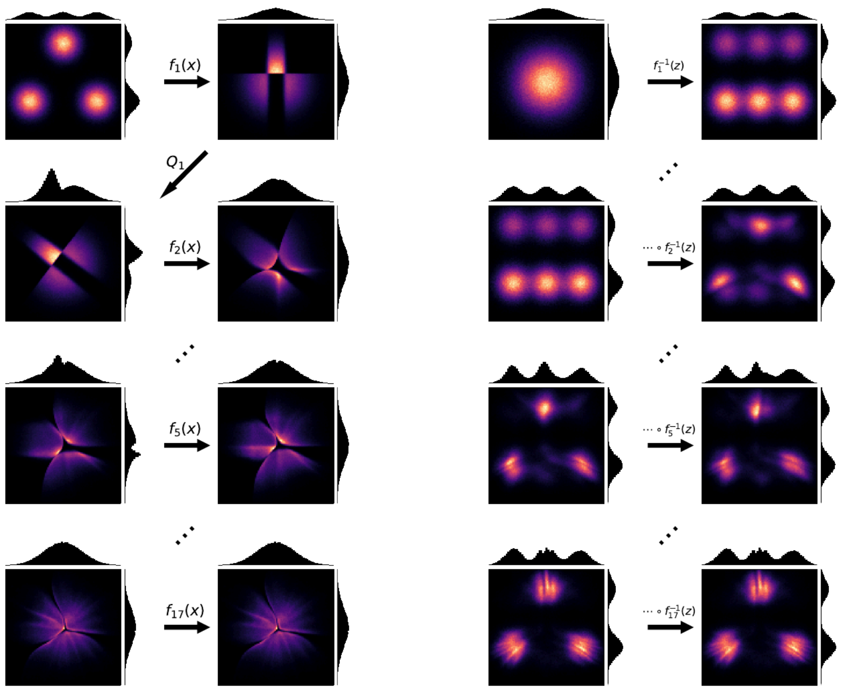


Figure 3: Gaussianization learns a mixture distribution with three modes. *(Left)* Forward process. $p(x)$ is transformed to $p(z)$ with iterative addition of layers. *(Right)* Reverse process. Image taken from [1].

This formulation has the significant advantage that the 1-dimensional transport to a normal distribution is already known analytically. Thus, for a single layer, we can solve for the optimal transform parameters s and t within a single step. This layerwise training entirely eliminates the need for backpropagation on a loss objective, such as the change of variables formula as shown in (6).

As a result, training the same number of layers is multiple orders of magnitude faster for Gaussianization than for Coupling Flows. However, the expressiveness of each individual layer is much lower, since unlike for Coupling Flows, dependencies between dimensions are not modelled at all or at most by the interaction between multiple layers for end-to-end training. We discuss this trade-off further in Section 3.

2.5 Continuous Normalizing Flows

Continuous Normalizing Flows (CNFs), as introduced by [24] and [25] are also trained using maximum likelihood, like their namesake. The fundamental formulation of the transform between distribution spaces differs, however. Where Normalizing Flows use a composition of functions, CNFs view the flow of probability as the drift of a fluid, parameterized by an ordinary differential equation (ODE):

$$\frac{\partial}{\partial t} x_t = f(x_t; t) \tag{13}$$

where f is the drift velocity field, which is learned by a neural network. An example is shown in Figure 4. Individual samples from the corresponding probability distributions can thus be viewed as particles in the fluid. We sample by integrating the ODE from $t_0 = 0$ (latent) to $t_1 = 1$ (data):

$$x_1 = x_0 + \int_0^1 f(x_t; t) \,\mathrm{d}t \tag{14}$$

The process is thus trivially invertible, since exchanging the integration bounds simply inverts the sign. To train the flow network, we use the continuous analogue of (6), the instantaneous change-of-variables formula:

$$\frac{\partial}{\partial t} \log p_t(x) = -\operatorname{tr} \frac{\partial f}{\partial x_t} \tag{15}$$

$$\log p_1(x) = \log p_0(z) - \int_0^1 \operatorname{tr} \frac{\partial f}{\partial x_t} \,\mathrm{d}t \tag{16}$$

where we now dub p_0, p_1 the probability distribution at time t_0, t_1, respectively. Note that this convention is opposite to that of Diffusion Models, where we have data at $t = 0$ and noise at $t = T$.

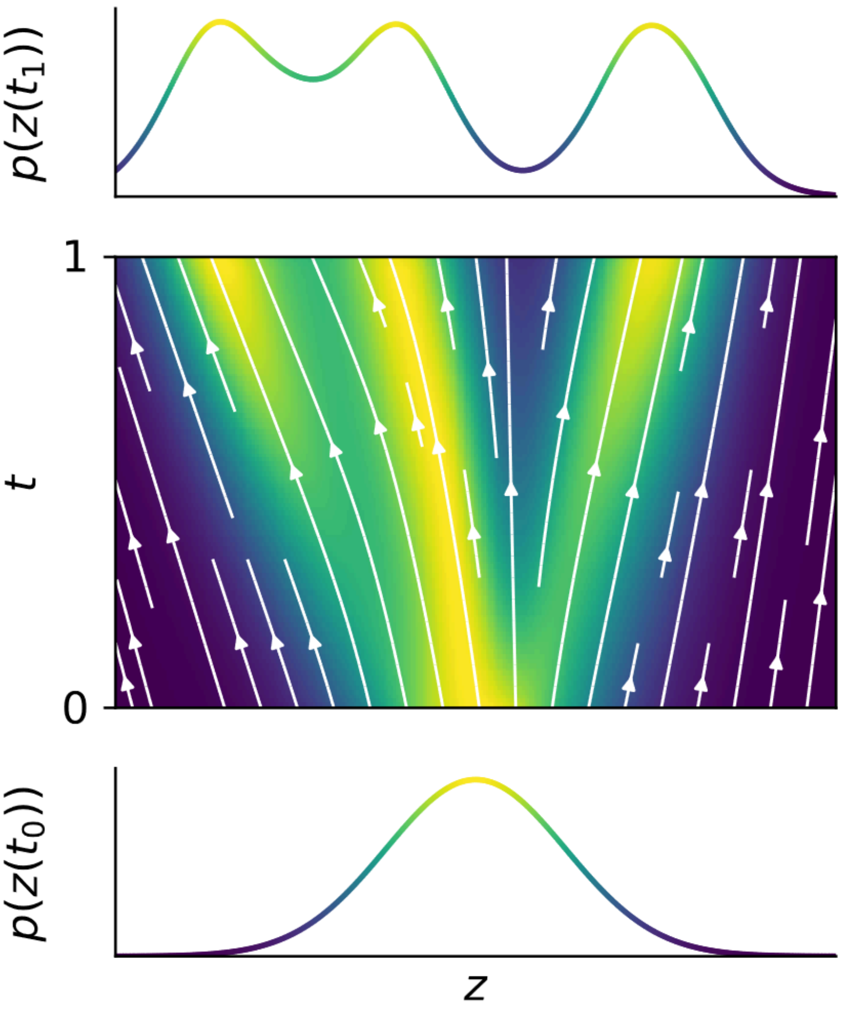


Figure 4: A CNF maps between a simple latent prior and a complex data distribution via an ODE. The probability flow field is shown in the center. Image taken from [25].

Maximising (16) is tractable since the Jacobian trace of $\operatorname{tr} \frac{\partial f}{\partial x_t}$ can be efficiently estimated with Hutchinson's algorithm [26]. The integration can be solved using any black-box ODE solver, such as the explicit Euler:

$$x_{t+\mathrm{d}t} = x_t + \frac{\partial x_t}{\partial t}\,\mathrm{d}t \tag{17}$$

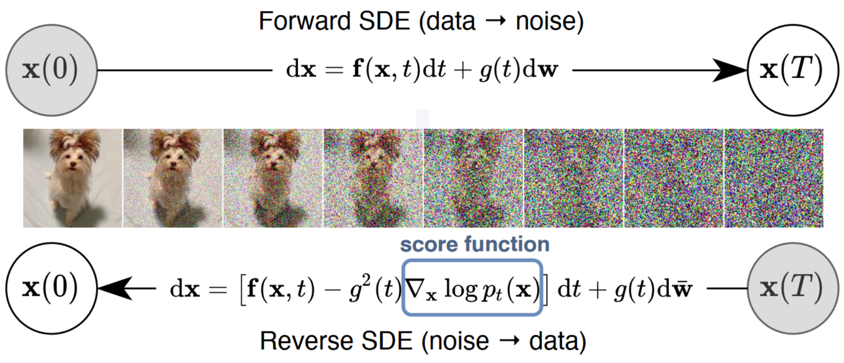
The primary advantage of this approach is that it is trivially invertible without architectural assumptions about f. Thus, the Jacobian J_f can be of any form, which allows for more expressive transforms than in Normalizing Flows and thus higher-quality distributional representations. Network parameters within f are also efficiently shared for multiple time points t.

However, simulation of the flow is expensive, typically requiring hundreds to thousands of network evaluations for a sufficiently small numerical error. Generating new data samples is thus multiple orders of magnitude slower than for single-step models. As such, training with the maximum likelihood objective in (16) can quickly become prohibitively expensive when using large networks. Furthermore, simulation-based training is often unstable due to the chaotic nature of ODEs and the resulting sensitivity of the integration to its initial conditions.

2.6 Diffusion Models

Particularly in high dimensions, transforming between distributions can be difficult for any of the aforementioned models. Training with maximum likelihood is either slow, unstable, or the models are not expressive enough, leading to artefacts in the learned distribution and thus mediocre samples.

Diffusion Models [27], [28] have quickly gained popularity in recent years due to their ability to overcome these difficulties. They are trained by first repeatedly adding controlled amounts of noise to data samples in order to transform them to the standard normal distribution. Thanks to the central limit theorem, any distribution can be transformed to the standard normal in this manner.



Forward SDE (data → noise)
$\mathbf{x}(0)$
$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w}$
$\mathbf{x}(T)$
score function
$\mathbf{x}(0)$
$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\boxed{\nabla_\mathbf{x} \log p_t(\mathbf{x})}\right] dt + g(t)d\bar{\mathbf{w}}$
$\mathbf{x}(T)$
Reverse SDE (noise → data)

Figure 5: Illustration of the forward and reverse processes of a Diffusion Model. The forward direction follows a standard Wiener-process SDE. The reverse process yields a score-based generative model. Image taken from [29].

The process is invertible if the amount of added noise in each step is sufficiently small. The Diffusion Model learns to remove these small amounts of Gaussian noise from samples until we arrive at the target distribution. Both the forward and reverse process are shown in Figure 5.

Diffusion Models are often also dubbed Score Matching, since the learned addition (or removal) of noise is equivalent to learning the score $\nabla \log p(x)$ of the distribution. Intuitively, this equality can be understood by considering the flow of a sample in the forward direction. Adding noise moves samples from regions of high density to regions of low density. Therefore, the mean movement of samples under the addition of noise reflects the gradient of the (log-) density.

The primary advantage of this training method is that since we effectively define a fixed forward process, we can remove the maximum likelihood loss altogether, and need not integrate all the way to the latent distribution to evaluate the loss. Instead, we can evaluate the network at just a single time step in the forward process, since jumping to any noise levels is trivial, given the target data sample. This allows us to directly regress to the fixed (denoised) solution. Therefore, this training method is usually called **simulation-free**.

However, while this makes training Diffusion Models fast, inference is usually expensive, requiring hundreds to thousands of network evaluations for high-quality samples, much like CNFs.

Recent work on Diffusion Models has focused on a significant reduction in the number of required network evaluations via distillation [30], as well as a different training scheme dubbed Consistency Models [31], [32].

2.7 Flow Matching

Flow Matching [10], originally dubbed Rectified Flow in [9], unifies the approaches of CNFs and Diffusion Models within a single model. Similar to CNFs, Flow Matching considers the flow of probability as a time-continuous ODE:

$$\frac{\partial}{\partial t} x_t = f(x_t; t) \tag{18}$$

Much like Diffusion Models, we also regress to a fixed forward process, thus gaining simulation-free training. Furthermore, Flow Matching generalises this idea to arbitrary distributions, leaving the notion of noisy diffusion behind. Instead, we regress f to the straight line interpolation between samples of the distribution:

$$\frac{\partial}{\partial t}x_t = \mathbb{E}[tx_1 + (1-t)x_0] \tag{19}$$

for all x_t on the interpolation line, and $x_0 \sim p_0, x_1 \sim p_1, t \sim \mathcal{U}(0,1)$. Points where the interpolation yields conflicting trajectories are averaged over in the expectation. The resulting flow is free of collisions and thus fulfils the continuity equation

$$\frac{\partial}{\partial t}p_t(x) + \nabla \cdot (p_t(x) f(x;t)) = 0 \tag{20}$$

For a visual representation, see Figure 6.

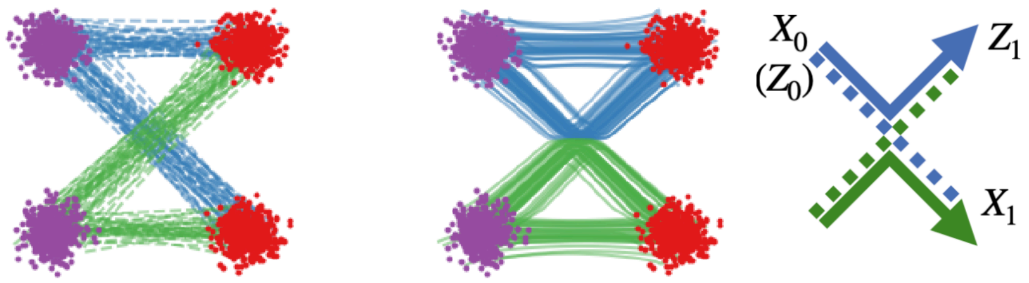


Figure 6: Averaging of Collisions in Flow Matching. *(Left)* Random assignment between the purple and red distributions yields conflicting trajectories, shown in blue and green. *(Center and Right)* The trained flow averages the velocity at collision points, yielding a flow trajectory without collisions that fulfills the continuity equation. Image taken from [9].

The absence of random noise within the flow integration for Flow Matching allows for very fast sampling, given that the integration trajectory is nearly straight. This becomes evident, particularly for a perfectly straight trajectory under constant velocity. In this case, the explicit Euler solves the ODE with zero error in a single function evaluation:

$$x_1 = x_0 + f(x_0; 0) \tag{21}$$

However, in order to achieve straight trajectories, collisions, as shown in Figure 6, must already be eliminated prior to drawing the interpolation in (19). The authors of [9] propose a method called ReFlow, which achieves this by replacing the dataset with a paired simulated dataset. The result is shown in Figure 7. However, this introduces a simulation error into later training runs. In Section 3, we show how this matching can be achieved without introducing such an error, using mini-batch Optimal Transport assignments instead.

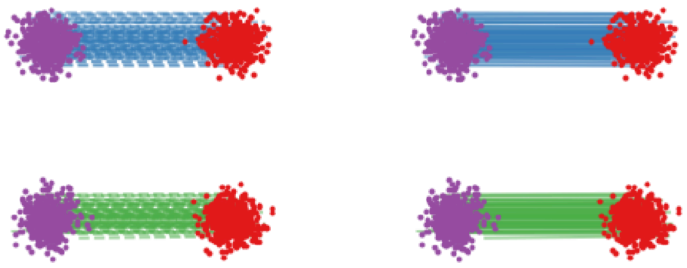


Figure 7: Straight Flow Matching. *(Left)* Optimal Assignment from data to latent space does not yield conflicting trajectories. *(Right)* The resulting flow is perfectly straight. Image taken from [9].

Recent other work on Flow Matching has focused on a controlled reintroduction of noise for its regularising effect [33]. Furthermore, Flow Matching is often leveraged as an efficient alternative to Diffusion Models, for instance in [12] and [34].

2.8 Model Overview

Model	Training Speed	Sampling Speed	Exact Likelihood	Free-Form Jacobian
Gaussianization	★	▲	✓	✗
Coupling Flows	▲	▲	✓	✗
CNFs	⬣	⬣	✓	✓
Diffusion Models	▲	⬣	✗	✓
Flow Matching	▲	●	✓	✓

★	▲	●	⬣	✓	✗
No Backprop	Single-Step	Few-Step	Many-Step	Yes	No

Table 1: Qualitative comparison overview of generative models.

3 Methods

In this section, we provide an overview of the existing and newly developed methods used to perform the experiments described in Section 4.

3.1 Convergence Rate of Gaussianization

This section is an adjusted version of parts of [1].

As described in Section 2.4.2, Gaussianization is a variant of Normalizing Flow that transforms data via a composition of 1-dimensional, unconditional functions. While this allows very fast training without backpropagation, it also reduces the expressiveness of each individual layer. Contrary to Coupling Flows, dependencies between dimensions are only modelled via the interaction of multiple layers.

We would like to investigate whether this is a worthwhile trade-off. Intuitively, one can understand that Gaussianization requires more layers than Coupling Flows to transform data to a standard normal. This becomes clearly evident when considering the data parameter covariance, as shown in Figure 8. For Coupling Flows, a scaling of $\Omega(1)$ has already been shown in [35].

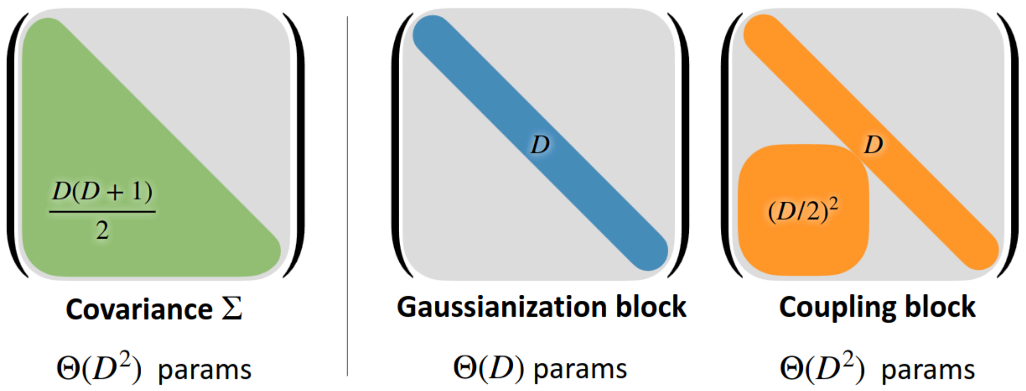


Figure 8: Parameter counting argument. The data covariance matrix has $\frac{D(D+1)}{2}$ degrees of freedom, of which Gaussianization can only learn D per layer. To transform the covariance to the unit matrix, Gaussianization should thus intuitively require $\Omega(D)$ layers. Coupling Flows can learn $D + \frac{D^2}{4}$ parameters per layer and thus only require a constant number of layers [35]. Image taken from [1].

In [1], we show an analytic derivation of the convergence rate of GAUSSIANIZATION for multivariate Gaussian distributions under random rotations. As part of this thesis, we further investigate the empirical scaling behaviour for arbitrary distributions, again under random rotations.

For these experiments, we choose the invertible function f as the rational-quadratic spline, which for each bin takes the functional form:

$$f(x) = \frac{\alpha_0 + \alpha_1 x + \alpha_2 x^2}{1 + \beta_1 x + \beta_2 x^2} \tag{22}$$

where, furthermore, α_i, β_i are constrained to yield a strictly monotonic spline basis function; see [19]. We use layerwise loss-free training as described in [22], fitting each 1-dimensional spline to the quantiles of the data, immediately yielding Optimal Transport to a standard normal within that dimension.

We determine the convergence scaling for GAUSSIANIZATION by first training a fixed number of layers L_{train} and then computing the number of layers L required to reduce the loss by a fixed ratio:

$$\gamma = \frac{\mathcal{L}'}{\mathcal{L}} < 1 \tag{23}$$

for a loss $\mathcal{L}$ bound from below by zero. We arbitrarily choose $\gamma = e^{-1} \approx 36.8\%$ as the target loss ratio, which is independent from the scaling with dimension. To determine L, we first observe that, in general, the loss follows a geometric series:

$$\mathcal{L} = \mathcal{L}_0 \gamma^{L_{\text{train}}} \tag{24}$$

Extrapolation of the geometric series allows us to predict the number of layers required for an arbitrary loss ratio:

$$L = \frac{\log(\mathcal{L}') - \log(\mathcal{L})}{\log(\gamma)} \tag{25}$$

Please refer to [1] for further details.

3.2 Context-Aware Learning

This section is an adjusted version of parts of [2].

Distribution shifts, as described in Section 2.2, often present a significant failure case for conventionally trained deep learning models. To achieve robustness, we choose to inform our model of such distributional shifts by providing it with environmental information.

In the setting of Domain Generalization, we already know that the distribution shift is associated with moving from one domain to another. Unfortunately, simply providing environmental information as a one-hot encoded domain vector is a feeble approach, as it requires that the exact number of possible environments is already known during training and that we can always exactly infer which environment an input originates from at inference time [2].

Consequently, this approach generalises poorly to real-world data, where environments can be constantly changing, novel environments can emerge, or the originating environment can be unknown entirely for some data points.

We instead choose to use a learned context embedding c from a set of n inputs $\mathcal{S}^{(n)}$ that originate from the same domain $\mathcal{D}_0$ as our singleton input x_0

$$c = E\left(\mathcal{S}^{(n)} = \{x_i \mid \mathcal{D}_i = \mathcal{D}_0\}_{i=1}^{n}\right) \tag{26}$$

where E is an encoder-network, called the set-encoder. The context c is then passed to an inference network alongside the singleton input x_0, which can be a classifier, regressor, or generative model, depending on the setting.

For the frameworks of classification and regression, [2] formalises an information-theoretic criterion necessary for this approach to yield a benefit over a baseline model that utilises only the singleton input x_0:

$$I\left(y; \mathcal{S}^{(n)} \mid x_0\right) > 0 \tag{27}$$

where we denote the mutual information as I and the target variable as y. We can break this criterion down into two weaker criteria:

$$I\left(\mathcal{D}_0; \mathcal{S}^{(n)} \mid x_0\right) > 0 \tag{28}$$

$$I(y; \mathcal{D}_0 \mid x_0) > 0 \tag{29}$$

Informally, these criteria have the following meanings for our approach:

1. $\mathcal{S}^{(n)}$ can improve our prediction for y
2. $\mathcal{S}^{(n)}$ can help us infer $\mathcal{D}_0$
3. $\mathcal{D}_0$ can improve our prediction for y

Note that (28) and (29) do not always directly imply (27); see [2] for a counterexample. The implication is almost always fulfilled for real-world data, however.

In Section 4, we highlight the importance of such contextual information by evaluating models under the phenomenon known in classical statistics as Simpson's Paradox. This paradox arises when several groups of statistical populations are not separated when drawing a trend. The trend reverses when groups are considered individually.

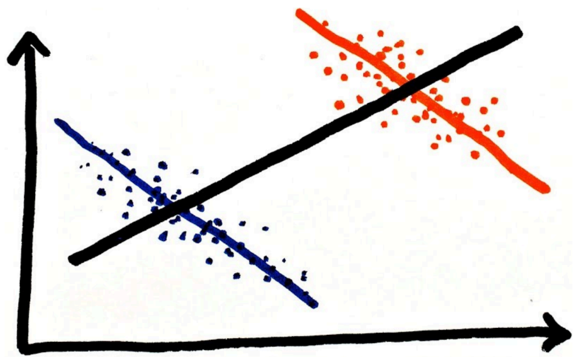


Figure 9: Simpson's Paradox. The global trend across the blue and orange groups is opposite that for each group, respectively. Image taken from [36].

To illustrate the paradox, consider the famous example from a real medical study [37] comparing the success rates of two treatment plans for kidney stones: one minimally invasive and another entailing open surgery. Open surgery presents higher success rates for both patients with small stones and large stones. Paradoxically, however, the minimally invasive treatment appears to be more successful when patients with small stones and large stones are grouped together.

	Minimally Invasive	Open Surgery	Total
Small Stones	87% (234 / 270)	**93% (81 / 87)**	88% (315 / 357)
Large Stones	69% (55 / 80)	**73% (192 / 263)**	72% (247 / 343)
Total	**83% (289 / 350)**	78% (273 / 350)	80% (562 / 700)

Figure 10: Simpson's Paradox for [37]. Without accounting for the severity of the patient's stone size, the minimally invasive treatment paradoxically appears as the better option, even though open surgery presents higher success rates for both stone sizes. Table data taken from [37].

A deep learning model that is informed only of the treatment type will naively learn the global trend, i.e., that the minimally invasive treatment is superior. Adding the stone size as contextual input to the network may help it group patients accordingly, thus overcoming the paradox.

3.3 Permutation-Invariant Neural Networks

This section is an adjusted version of parts of [2].

Set inputs are naturally permutation-invariant, meaning they follow the same joint probability under an exchange of elements:

$$p(x_1, x_2, ..., x_n) = p\left(x_{P(1)}, x_{P(2)}, ..., x_{P(n)}\right) \tag{30}$$

where $P : \mathbb{N} \to \mathbb{N}$ denotes an arbitrary permutation of indices with $n \in \mathbb{N}$ [2]. This can be intuitively understood by recognising that swapping the positions of two particles in a cloud of identical particles yields the very same cloud; see also Section 2.3.

Permutation-invariant neural networks thus present an architectural choice with a particularly favourable inductive bias for such inputs. Perhaps the simplest method to build such a network is to use the functional form of the sum-decomposition:

$$E\left(\mathcal{S}^{(n)}\right) = \rho\left(\sum_{i=1}^{n} \sigma(x_i)\right) \tag{31}$$

where ρ and σ can be any functions, e.g., neural networks. It is immediately apparent that E must be permutation-invariant since

1. σ acts equally on all x_i and is thus *permutation-equivariant*.
2. The summation $\sum_{i=1}^{n}$ is inherently **permutation-invariant**.
3. ρ acts only on an already **permutation-invariant** summary input.

Of course, any *permutation-equivariant* function may be chosen in place of the element-wise σ, as well as any **permutation-invariant** pooling in place of summation. Finding optimal pooling functions can be application-dependent and is still an active area of research. For instance, [38] propose using (induced) self-attention without positional encodings; [39] and [40] introduce a TopK pooling operator; and [41] use stacked sum-decompositions with equivariant poolings.

Our specific data pipeline differs by experiment and can thus be found in Section 4.

3.4 Optimal Transport Flow Matching

The ReFlow step, as proposed in the original paper [9], introduces a simulation error into the straightening of trajectories, where the quality of training data diminishes with each step of ReFlow.

We propose an algorithm that eliminates this simulation error by matching data and latent space exactly via Optimal Transport, thus retaining the original training data between steps of ReFlow.

First, we observe that for a given data point x_1, not all latent points x_0 are equally likely targets. In fact, we can choose the assignment $p(x_0|x_1)$ arbitrarily, so long as the latent marginal stays intact:

$$p(x_0) = \int p(x_0|x_1)p(x_1)\,\mathrm{d}x_1 \tag{32}$$

In order to obtain a first estimate for $p(x_0|x_1)$, we transform x_1 into the latent space, retrieving $\hat{x}_0$. This can be done with a few-step approximation of the flow. We can then weight a set of samples from the latent space with a Gaussian kernel, using their Euclidean distances to $\hat{x}_0$:

$$W_{ij} = \exp\left(-\frac{\| \hat{x}_0^{(i)} - x_0^{(j)} \|_2^2}{2\sigma}\right) \tag{33}$$

Where W_{ij} replaces $p(x_0|x_1)$ in the case of a finite number of samples. However, in order to fulfil (32), we need to assign each data point to each latent point exactly once, on average. This is true if and only if W is doubly stochastic, i.e., if W sums to 1 along both axes:

$$\sum_i W_{ij} = \sum_j \left(W_{ij}\right)^T = \mathbb{1} \tag{34}$$

which we can achieve by repeatedly normalising W with a softmax.

```python
import torch
from torch import Tensor

def match(x0: Tensor, x1: Tensor, sigma: float, batch_size: int, steps: int):
    x0 = list(torch.split(x0, batch_size))
    x1 = list(torch.split(x1, batch_size))

    for i in range(len(x0)):
        pairwise_distances = torch.cdist(x0[i], x1[i])
        W = torch.exp(-0.5 * pairwise_distances ** 2 / sigma)

        for _ in range(steps):
            W = torch.softmax(W, dim=0)
            W = torch.softmax(W, dim=1)

        permutation = torch.multinomial(W, 1)
        permutation = permutation.squeeze(1)

        x1[i] = x1[i][permutation]

    return torch.cat(x0), torch.cat(x1)
```

Listing 1: Our proposed mini-batch Optimal Transport algorithm.

Our proposed algorithm is equivalent to the Sinkhorn-Knopp Optimal Transport algorithm [42], [43], where a negative cost matrix is repeatedly normalised to yield a doubly-stochastic transport plan. A log-stabilised version can be found in Appendix A.

It is important to minimise σ in order to achieve useful one-to-one matchings that optimise the trajectory straightness. However, the initial predictions $\hat{x}_0$ of an untrained network may not suffice for a good matching. As such, we start with $\sigma = 1$ and anneal σ over training towards zero, which naturally straightens the flow trajectories throughout training.

Computing the assignment matrix W_{ij} is of quadratic complexity $\mathcal{O}(n^2)$ with n the number of items to be matched. In order to make the algorithm tractable, we propose computing W within mini-batches, reducing the complexity to $\mathcal{O}\left(\frac{n}{b}b^2\right) = \mathcal{O}(nb)$ for batch size b.

Note that simultaneously, similar algorithms were independently developed in [44] and [45] where W is instead computed with Euclidean distances measured directly between data and latent space. Similar to our proposed algorithm, [44] uses the Sinkhorn-Knopp algorithm. [45] proposes using an optimal one-to-one assignment with the Hungarian algorithm [46].

Computing the distances directly between data and latent space has the significant advantage that the data need not be simulated into the latent space. The network instead learns a continuous surrogate function for the respective Optimal Transport matching algorithm. However, the theoretical properties of optimising Euclidean costs between high-dimensional data and latent spaces remain unexplored, as these are often subject to the curse of dimensionality. Furthermore, [44] mentions that in high dimensions, the mini-batch approximation for the matching may incur an error for the Optimal Transport cost.

Due to its improved computational efficiency, we will use the Optimal Transport matching algorithm as proposed in [44] in Section 4.

4 Experiments

4.1 Empirical Analysis of the Convergence Rate of Gaussianization

This section is an adjusted version of parts of [1], which covers a theoretical analysis of the convergence rate of Gaussianization with random rotations for Gaussian data in dependence of the data dimension D, as well as an empirical extension of the theoretical result to arbitrary distributions.

We present a selection of experiments from the paper, to which the author of the thesis contributed significantly. For the theoretical results as well as additional experiments, please refer to the paper.

4.1.1 Banana Dataset

Dataset: We first consider the convergence rate on an artificially constructed autoregressive toy dataset, built specifically to create a controlled number of dependencies between dimensions:

$$p(x) = p(x_1) \prod_{i=2}^{D} p(x_i | A_i) \tag{35}$$

where the set A_i collects the random variables that x_i depends upon. Specifically, we consider three scenarios, with varying degrees of dependencies between dimensions:

1. Every variable depends on every previous variable:

$$A_i^{(1)} = \{x_1, ..., x_{i-1}\} \tag{36}$$

2. Only a core set of d variables depends on all previous variables. The remaining variables only depend on the core set:

$$A_i^{(2)} = \begin{cases} A_i^{(1)} & \text{if } i \leq d \\ A_d^{(1)} & \text{if } i > d \end{cases} \tag{37}$$

3. Same as the previous case, except the remaining variables are independent Gaussian noise:

$$A_i^{(3)} = \begin{cases} A_i^{(1)} & \text{if } i \leq d \\ \emptyset & \text{if } i > d \end{cases} \tag{38}$$

The dependencies are introduced via the following function, giving the dataset its characteristic banana-shaped density mode:

$$m_i(A_i) = m_0 + 5 \tanh\left(\frac{1}{10} \sum_{x_j \in A_i} s_{ij} x_j^2 \right) \tag{39}$$

See [1] for further details on the data-generating process.

Pipeline: For all experiments in this section, we apply unconditional GAUSSIANIZATION with random rotations, as shown in Figure 3. As shown in Section 3, we choose rational-quadratic splines for the 1-dimensional transforms. We train a fixed number of layers and compute the convergence rate as shown in (25).

Results: Shown in Figure 11. The theoretically expected linear scaling is confirmed for cases 1 (36) and 3 (38). For case 2 (37), approximately a constant number of layers is sufficient, particularly when the core set of variables is small ($d \ll D$).

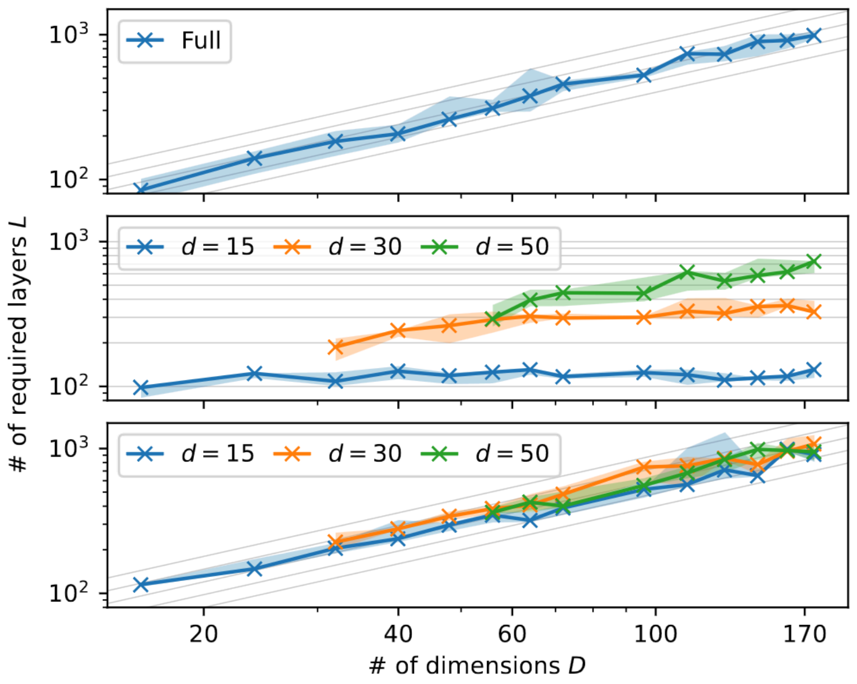


Figure 11: The number of required layers for GAUSSIANIZATION on toy data computed for $\gamma = 36.8\%$ as shown in (25). *(Top)* If all dimensions depend on one another, we recover the expected linear scaling. *(Middle)* When the trailing dimensions $i > d$ depend on the core set, we can only recover a weak scaling. Particularly for small core sets $d \ll D$, the scaling is approximately constant. *(Bottom)* When the trailing dimensions $i > d$ are independent Gaussian noise, we again recover the linear scaling. Image taken from [1].

4.1.2 EMNIST

Dataset: In order to estimate the convergence scaling for real-world datasets, we consider the EMNIST dataset [47], which presents an extension of the standard MNIST to handwritten letters. We construct variants of the dataset with a differing number of dimensions by scaling the images between 2×2 pixels and the original 28×28, see Figure 12.

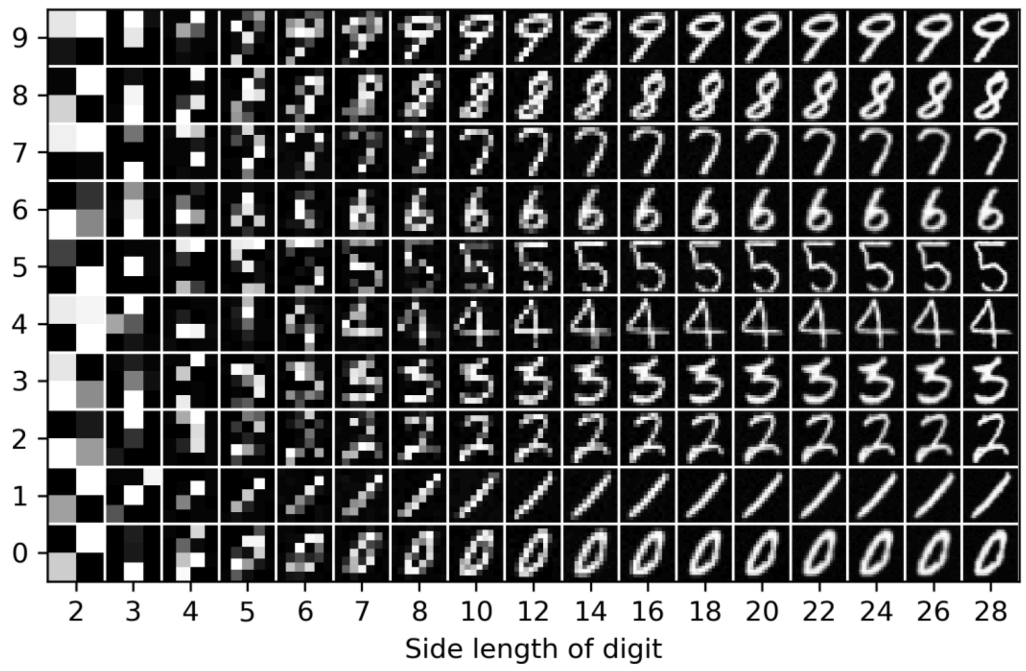


Figure 12: Our multi-scale EMNIST dataset. Image taken from [1].

Results: Shown in Figure 13. We recover the expected linear scaling for low-dimensional variants of the dataset, up to 10×10 pixel images, i.e. $D \approx 100$. Beyond this, the number of required layers saturates, presumably because, for high resolutions, most pixels depend largely on their neighbours. This corresponds to the toy experiment case (37), where we also found an approximately constant scaling for $d \ll D$.

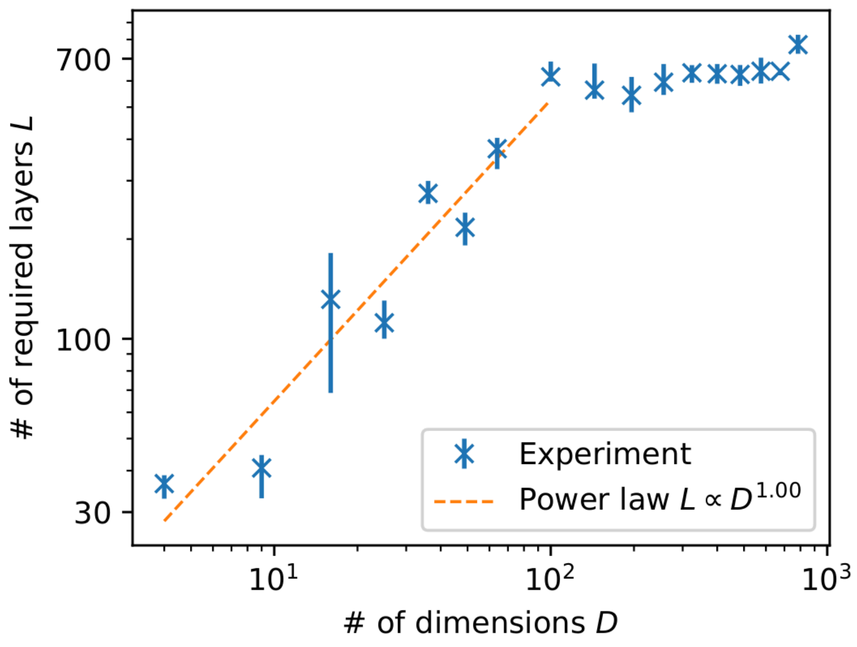


Figure 13: Empirical scaling of required layers for GAUSSIANIZATION on our multi-scale EMNIST dataset. Images of higher resolution require more layers. The scaling is linear up to 10×10 pixel images, i.e., $D \approx 100$, and saturates after that. We presume that this is because the digits are fully recoverable at that point, and pixels start to depend largely on their neighbours. Data points show the median result, and error bars cover 90% of the training runs. Image taken from [1].

4.2 Context-Aware FLOW MATCHING for ModelNet10 Point Clouds

Dataset: As an initial toy experiment, we train a generative model on 3-dimensional point clouds, sampled on the surfaces of furniture from the ModelNet10 dataset [48]. The dataset consists of close to 5000 furniture items across 10 categories.

It has been used as a typical benchmark for generative models; see, for instance, [11] and [12]. The dataset allows us to sample point clouds of arbitrary cardinality. However, in order to stay comparable to the related work, we choose to use a cardinality of 2048 points in training. Samples from the dataset can be seen in Figure 14.

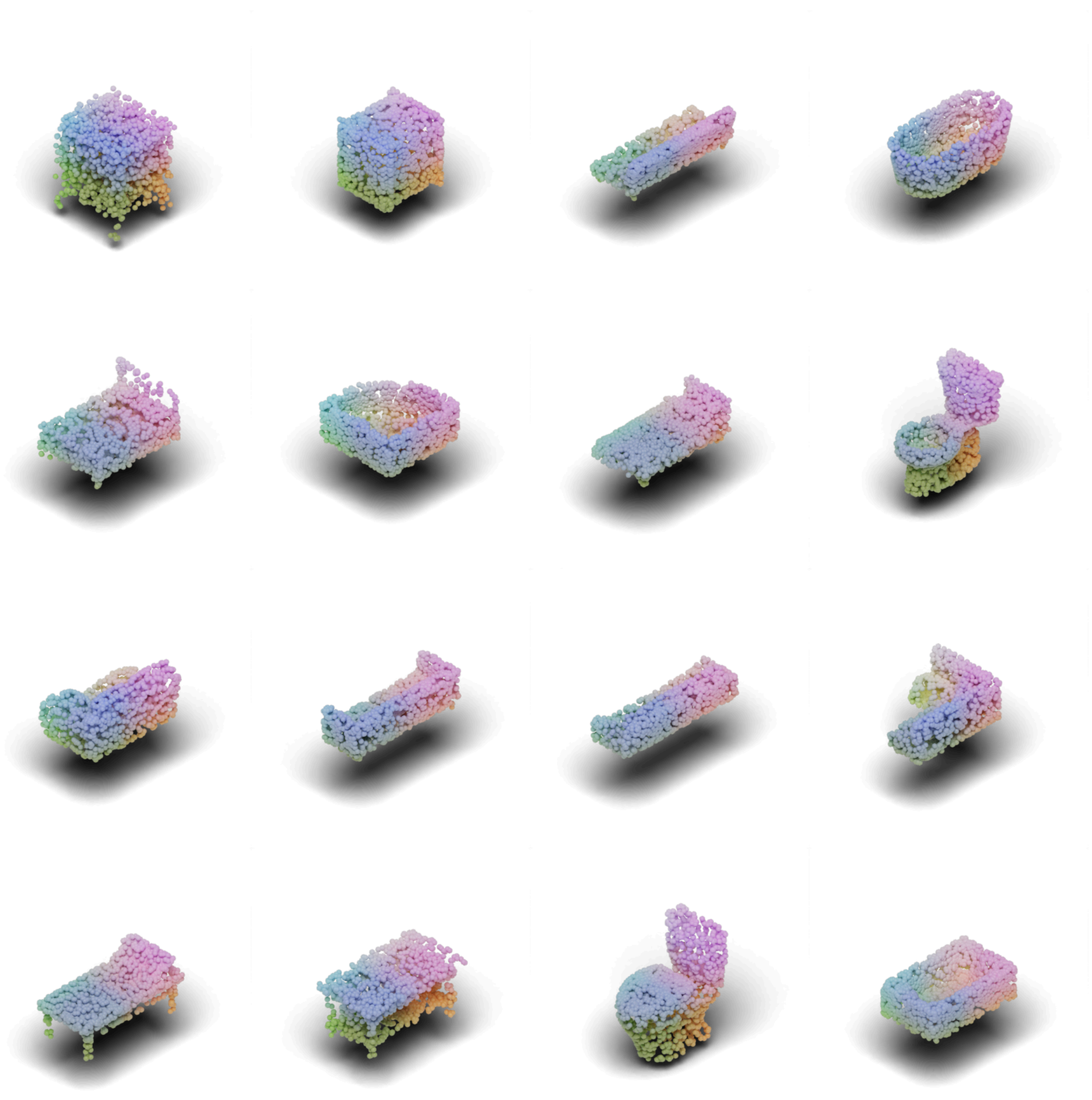


Figure 14: Example shapes generated by sampling points on furniture meshes from the ModelNet10 dataset [48]. Image rendered with BLENDER-PLOT [49].

Pipeline: We employ permutation-invariant networks as well as Optimal Transport FLOW MATCHING for this experiment, similar to [12].

The task of the set-encoder is to learn a contextual embedding for the shape of the furniture item, while the flow conditionally transports each point within the cloud to a standard normal distribution.

Importantly, unlike [12], we do not apply voxelization, with the intent that our set-encoder architecture may generalise to higher-dimensional set-based data for later experiments. This is not possible with voxelization and convolution, since these techniques become prohibitively expensive for high-dimensional data (complexity is typically exponential).

As another significant difference to [12], we apply the Optimal Transport matching procedure described in [44] in order to straighten flow trajectories. Note that we match to a standard normal *within each set*, such that the latent distribution of the flow is always a standard normal *given the context embedding* c. Since the cardinality of the set is high (2048), Optimal Transport matching can become prohibitively expensive. To avoid this, we sample a subset (128) in each batch when training the flow.

An illustration of the architecture is shown in Fi

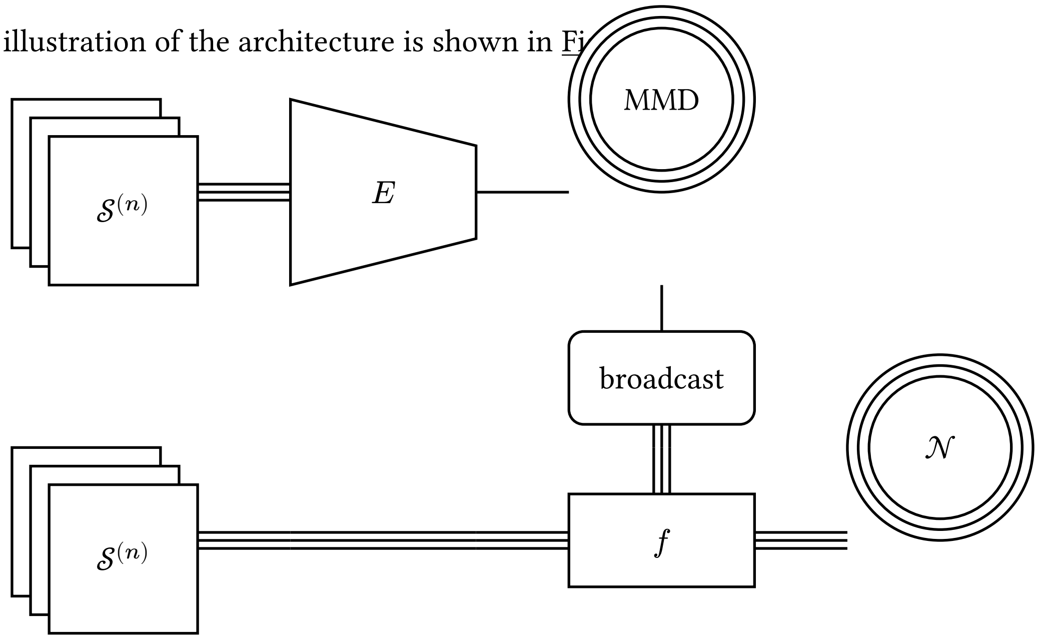


Figure 15: Context-Aware Flow Matching Architecture. The set input $\mathcal{S}^{(n)}$ is encoded into a single contextual embedding by the set-encoder E. The set is also passed element-wise to the flow network f, which transports it to a standard normal $\mathcal{N}$ conditioned on the contextual embedding. The contextual embedding is broadcasted and concatenated to every flow input. The latent distribution of the set-encoder can be encouraged to follow a standard normal distribution via an MMD loss term. This enables sampling contextual embeddings at inference time.

<u>Loss:</u> We use a linear combination of the standard loss for Optimal Transport Flow Matching, as well as the Maximum Mean Discrepancy (MMD) to encourage the latent distribution for the set-encoder to follow a standard normal. To balance the two losses, we introduce a hyperparamer $\gamma \in [0, 1]$.

$$\mathcal{L}\left(\mathcal{S}^{(n)}\right) = \gamma \mathcal{L}_E\left(\mathcal{S}^{(n)}\right) + (1 - \gamma)\mathcal{L}_f\left(\mathcal{S}^{(n)}\right) \tag{40}$$

Where $\mathcal{S}^{(n)}$ is the set input.

$\mathcal{L}_f$ is the standard loss for Optimal Transport Flow Matching [9], [10], [44]:

$$\mathcal{L}_f\left(\mathcal{S}^{(n)}\right) = \mathbb{E}_{(x_0,x_1)\sim\pi, t\sim\mathcal{U}(0,1)}[\mathrm{MSE}(f(tx_1 + (1-t)x_0; t, c), (x_1 - x_0))] \quad (41)$$

with f the flow network, t the integration time point, c the contextual set-encoder output, and π the Optimal Transport matching plan $\pi = \mathrm{sinkhorn}\left(\mathcal{S}^{(n)}, \mathcal{X}^{(n)} \sim \mathcal{N}(0, \mathbb{I})\right)$ as a joint distribution between data and latent space.

Finally, $\mathcal{L}_E$ is the set-encoder loss:

$$\mathcal{L}_E\left(\mathcal{S}^{(n)}\right) = \mathbb{E}_{x\sim\mathcal{N}(0,\mathbb{I})}\left[\mathrm{MMD}\left(E\left(\mathcal{S}^{(n)}\right), x\right)\right] \quad (42)$$

Setting $\gamma \to 0$ places emphasis on the reconstruction quality, but sampling may suffer when γ is chosen too small (i.e., the set-encoder latent distribution is not normal enough). Setting $\gamma \to 1$ places emphasis on the encoder generating more normally distributed codes, but it may destroy information in the latent code if γ gets too large, leading to both poor sampling and reconstructional quality. We recommend setting $\gamma = 0.5$ for balanced training, but smaller values for γ can be favourable if set-based data is available at inference time. γ can also be set to normalise the loss gradient norm at the start of training:

$$\gamma_{\text{init}} = \frac{\|\nabla_\theta \mathcal{L}_f\|}{\|\nabla_\theta \mathcal{L}_f\| + \|\nabla_\theta \mathcal{L}_E\|} \approx \frac{\mathcal{L}_f}{\mathcal{L}_f + \mathcal{L}_E} \quad (43)$$

and then annealed towards 0.5 during training. This is helpful if training is initially unstable.

Results: Qualitative samples can be seen in Figure 16 and Figure 17.

Similar to [12], we also interpolate between samples in the latent space of the encoder in Figure 18. Importantly, this figure can highlight whether the resulting model is overfit. Random samples from an overfit model will appear better, but this interpolation will not change smoothly with interpolation time when the flow is only memorising the training data.

In general, our qualitative samples look convincing, and our model does not appear to have overfit. The reconstructed samples are nearly indistinguishable from the dataset. Our random samples are slightly noisier than the existing state-of-the-art, likely due to our simplified architecture.

Some furniture items in the dataset are also already rotated with respect to one another, which explains the noise in the armrests of some chairs and sofas. In general, conditioning on the furniture type, either with a one-hot encoding or text-guided embeddings, training one model per furniture type, as well as training on a larger dataset (e.g. ModelNet40 [48]) may yield significant improvements.

However, we opt not to focus on improving this model any further due to the aforementioned reason: generalising to other datasets.

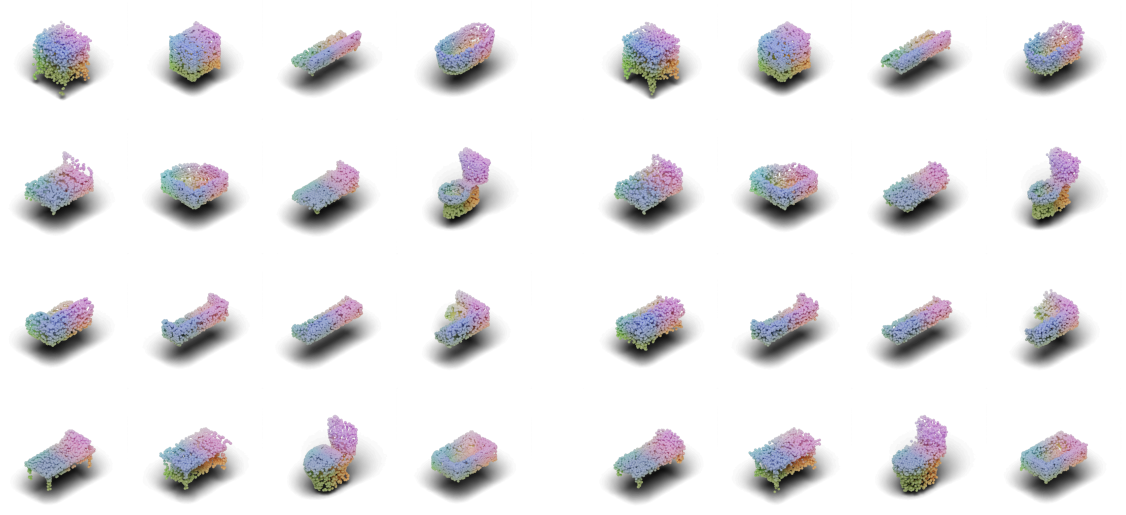


Figure 16: *(Left)* Random samples from the validation set. *(Right)* Model reconstructions of the samples. Images rendered with BLENDER-PLOT [49].

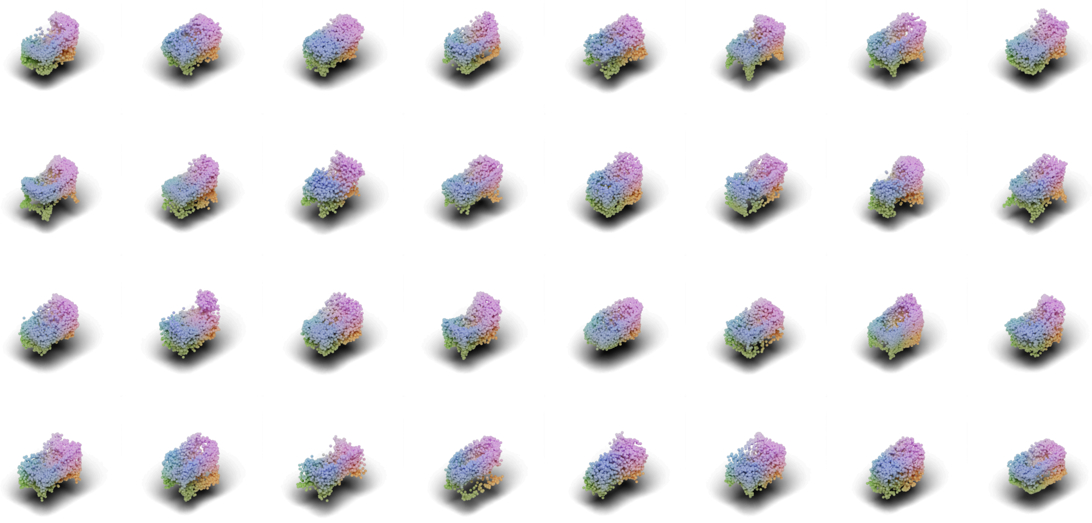


Figure 17: Random samples from the trained model. Image rendered with BLENDER-PLOT [49].

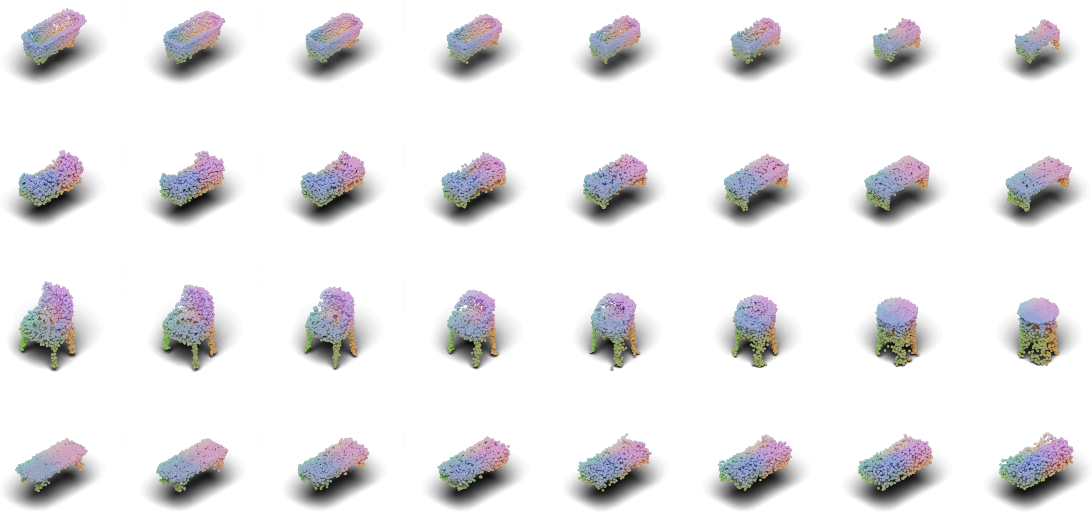


Figure 18: Sample interpolation on the validation set. Our architecture allows a smooth transition between shapes by linearly interpolating between context embeddings in the latent space of the set-encoder. Images rendered with BLENDER-PLOT [49].

[2]https://larskue.github.io/context-aware-flow-matching/

We further encourage you to visit the project page[2] to see a visualization of the learned manifold in a video where we rotate the latent space of the flow, as well as a visualization of the data generating process.

4.3 Context-Aware Domain Generalization

This section is an adjusted version of parts of [2], which covers a novel approach to Domain Generalization using model selection via set-based context-informed environment detection, as well as a formalisation of theoretical criteria for when this approach yields a benefit (see also (27) - (29)).

We present a selection of experiments from the paper, to which the author of the thesis contributed significantly. For additional experiments and further details, please refer to the paper.

4.3.1 Simpson's Paradox: Domain Classification

Dataset: In order to highlight the effect of the set size on the power of the contextual embedding, we first evaluate a synthetic toy dataset that simulates Simpson's paradox as described in Section 2. The dataset consists of a mixture of 2D multivariate normal distributions, where the first dimension is used as a feature and the second will later be used as a regression target. Each component of the mixture represents a separate domain, which is the classification target for the current experiment.

The mixture components are chosen to lie on a trend line that is opposite to the trend within each mixture, thus inherently enabling Simpson's paradox as described in Section 3. We achieve this by using a negative global trend and choosing the covariance matrix of each mixture component as a scaled and rotated identity matrix with a positive trend.

We vary the distance between component means in order to control the overlap between domain marginals. This effectively controls the mutual information between the set and the domain, given the singleton sample. When the marginals strongly overlap, the singleton sample is uninformative of the domain, which increases the relative importance of the set.

An illustration of the dataset for a spacing value of 2.0 can be seen in Figure 19. The training data contains 10k samples for each domain.

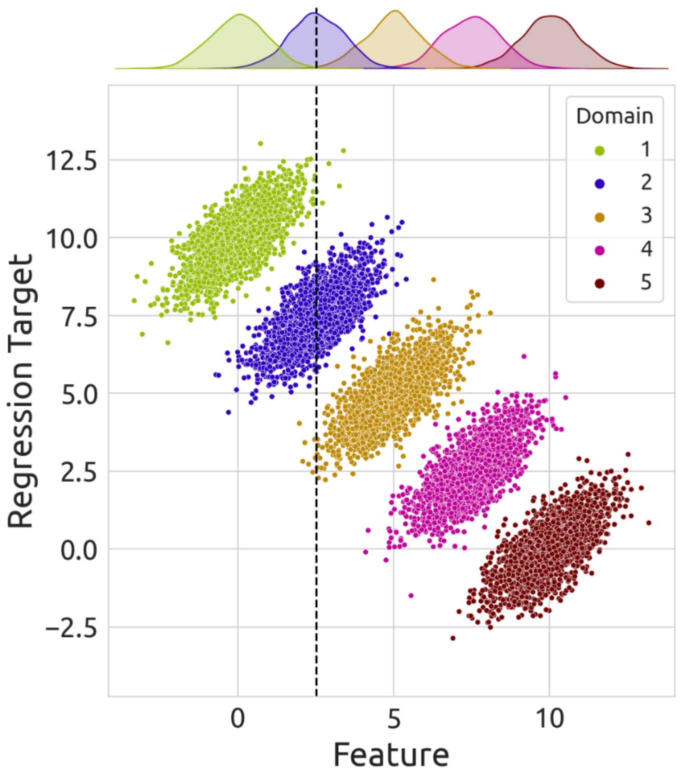


Figure 19: Simpson's paradox toy dataset, with domain marginal distributions shown at the top. Without environmental information, the marked input at $x = 2.5$ could belong to any of the domains numbered 1, 2, or 3. Image taken from [2].

Pipeline: We first evaluate a fully connected, mean-pooled set-encoder with 5 layers and compare it to a fully connected baseline network (with no pooling) of equal size. Both models aim to predict the domain $\mathcal{D}_0$ of the set-input $\mathcal{S}^{(n)}$ or singleton input x_0, respectively. The set-summary output of the set-encoder is used directly as the logits for classification. As such, with this pipeline, we directly compare the informational power of the set and the singleton sample.

Loss: As is typical for classification, we employ the cross-entropy loss H:

$$\mathcal{L}(x_0) = H(f(x_0), \mathcal{D}_0) \tag{44}$$

$$\mathcal{L}\left(\mathcal{S}^{(n)}\right) = H\left(E\left(\mathcal{S}^{(n)}\right), \mathcal{D}_0\right) \tag{45}$$

where f is the baseline network and E the set-encoder. See Appendix A for a detailed implementation.

Results: Shown in Figure 20. The set size has a significant impact on the classification accuracy. Even for small set sizes, the set-encoder significantly outperforms the baseline. As expected, the difference is particularly high when the distance between domains is small but not zero.

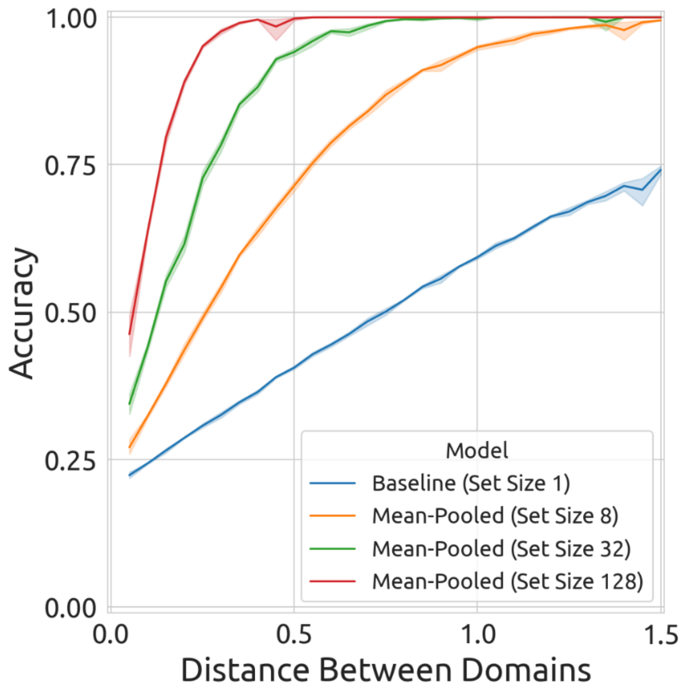


Figure 20: Domain classification accuracy by set size and domain distancing. Image taken from [2].

4.3.2 Simpson's Paradox: Out-of-Distribution Regression

Dataset: We use the toy dataset as shown in Figure 19, with one of the five domains chosen as out-of-distribution. The models are trained only on in-distribution data; the OOD domain is entirely unseen at inference time. Instead of classifying the domain, we now regress to the second dimension, given the first.

Pipeline: As shown in [2], for this experiment and the following ones, we apply a fully connected, mean-pooled set-encoder together with a fully connected regressor network. An illustration of the data pipeline can be seen in Figure 21. Since the problem is linear, we only use single-layer linear inference networks.

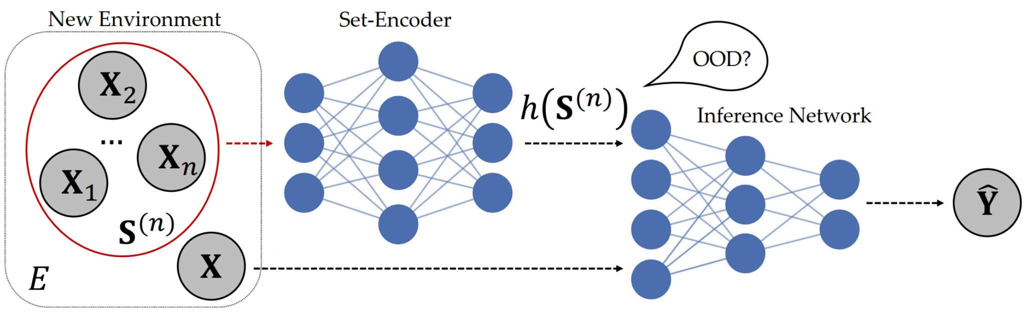


Figure 21: Data pipeline and architecture for context-aware Domain Generalization. Image taken from [2].

Loss: We use the mean squared error, as is standard for regression problems:

$$\mathcal{L}(x, \mathcal{S}^{(n)}; y) = \text{MSE}(f(x, E(\mathcal{S}^{(n)})), y) \tag{46}$$

Results: In Figure 22, we show that the baseline is unable to model the data-generating process, following the undesirable global trend instead. The set-based model is able to correctly resolve the paradox by recognising the domain-specific variance, even when required to extrapolate.

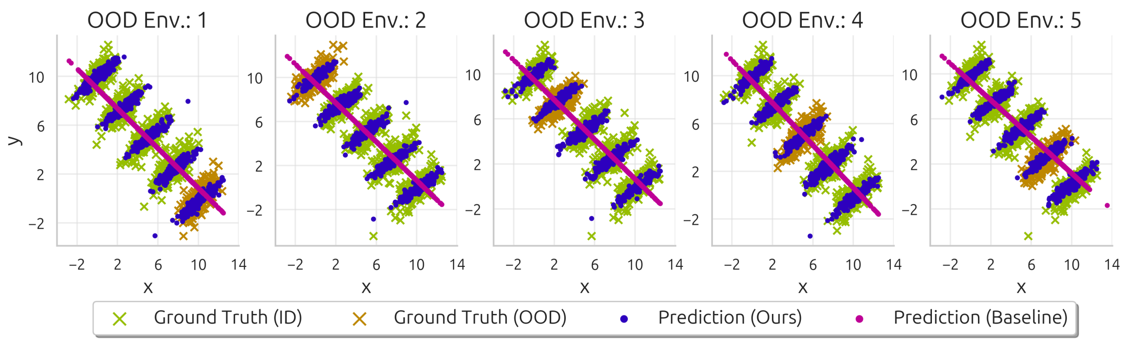


Figure 22: Linear model predictions for the dataset illustrated in Figure 19 for different ID and OOD settings. The set-based model is able to recognise the domain-specific variance, whereas the baseline only follows the global trend. Image taken from [2].

4.3.3 ColoredMNIST

Dataset: Following up on the low-dimensional toy experiments, we apply our approach to the ColoredMNIST dataset [50]. This is a variant of the MNIST dataset with two class labels: 0 for digits < 5 and 1 for digits ≥ 5. In 25% of images, the label is flipped, so a model relying purely on the shape of the digit will only achieve 75% accuracy.

Furthermore, for domains we consider in-distribution, there is a strong association between the background colour and the class label: 90% and 80% respectively. A baseline model trained on in-distribution data would thus learn to prefer this association over the shape association, to maximize accuracy. However, on out-of-distribution data, this approach fails, as the association is here only 10% - significantly worse than random guessing.

We consider this dataset ideal to showcase our approach since it features the typical trade-off between relying on domain-specific data attributes and learning an invariant but robust relationship.

Results: Shown in Table 2. As expected, the baseline model fails dramatically on out-of-distribution data. The invariant model performs similarly well in both domains but achieves suboptimal accuracy for in-distribution data, due to its inability to exploit the domain-specific background colour association. Our approach yields the best of both worlds, achieving near-optimal accuracy both for in- and out-of-distribution domains.

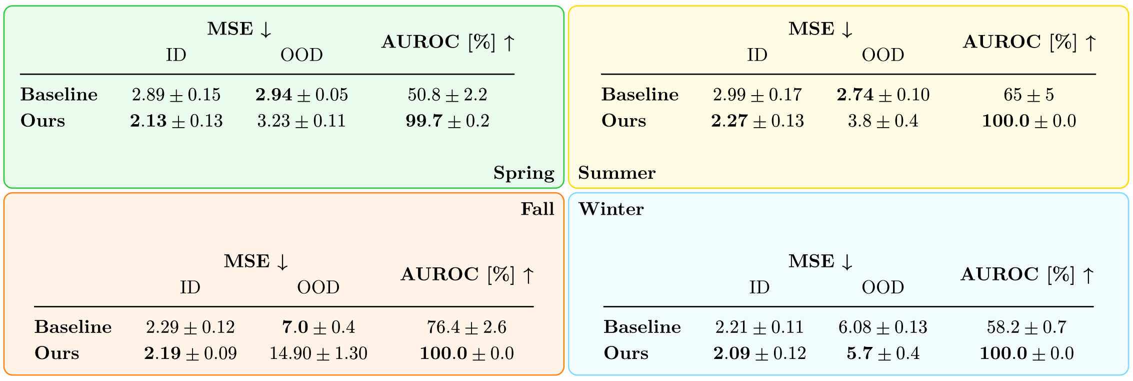
	Accuracy [%] ↑	
	In-Distribution	Out-of-Distribution
Baseline	**84.6** $\pm$ 0.3	10.2 $\pm$ 0.3
Invariant	72.8 $\pm$ 0.9	**73.1** $\pm$ 0.2
Selection (Ours)	**84.1** $\pm$ 0.3	**73.1** $\pm$ 0.2
Selection (Baseline)	**84.0** $\pm$ 0.3	14.0 $\pm$ 0.4
Bayes Optimal Classifier	**85.0**	**75.0**

Table 2: Mean and standard deviation of accuracy percentages for the ColoredMNIST dataset [50]. The standard deviation is derived from 5 runs using different seeds for data partitioning. The features extracted by our model allow for improved OOD detection compared to the features of the baseline model. Thus, our model can perform a favourable selection between the baseline model in the ID setting and the invariant model in the OOD setting. Table data taken from [2].

4.3.4 Failure Case Detection in BikeSharing

Dataset: Distribution shifts that require significant extrapolation may still present a failure case for our approach. Here, both the baseline model and our model are expected to experience a degradation in performance. We highlight this with the BikeSharing dataset [51], which contains over 17000 hourly counts of bike rentals between 2011 and 2012 within the Capital bikeshare system in Washington, DC. Here, we choose different seasons to represent different domains and evaluate our approach by training on all seasons, except one.

Results: Shown in Table 3. Our approach performs slightly superior to the baseline on in-distribution data, which suggests the domain information is helpful and not inferable from a single sample.



Spring

	MSE ↓		AUROC [%] ↑
	ID	OOD	
Baseline	2.89 ± 0.15	$\mathbf{2.94} \pm 0.05$	50.8 ± 2.2
Ours	$\mathbf{2.13} \pm 0.13$	3.23 ± 0.11	$\mathbf{99.7} \pm 0.2$

Summer

	MSE ↓		AUROC [%] ↑
	ID	OOD	
Baseline	2.99 ± 0.17	$\mathbf{2.74} \pm 0.10$	65 ± 5
Ours	$\mathbf{2.27} \pm 0.13$	3.8 ± 0.4	$\mathbf{100.0} \pm 0.0$

Fall

	MSE ↓		AUROC [%] ↑
	ID	OOD	
Baseline	2.29 ± 0.12	$\mathbf{7.0} \pm 0.4$	76.4 ± 2.6
Ours	$\mathbf{2.19} \pm 0.09$	14.90 ± 1.30	$\mathbf{100.0} \pm 0.0$

Winter

	MSE ↓		AUROC [%] ↑
	ID	OOD	
Baseline	2.21 ± 0.11	6.08 ± 0.13	58.2 ± 0.7
Ours	$\mathbf{2.09} \pm 0.12$	$\mathbf{5.7} \pm 0.4$	$\mathbf{100.0} \pm 0.0$

Table 3: Performance comparison between our model and the baseline on the BikeSharing dataset [51], broken down by target domain. We compare their performance in the ID and OOD settings (MSE), as well as their capability to detect a novel environment (AUROC). Both models fail in the OOD setting, but our model can detect with strong certainty when this is the case. We present the mean and standard deviation derived from 5 runs using different seeds for partitioning the data into training, validation, and test sets. Image taken from [2].

However, as expected, both the baseline and our approach suffer from performance degradation on out-of-distribution data. Fortunately, our approach is able to consistently detect this novel environment as a failure case, with an AUROC very close to 100% in all cases.

5 Conclusion

Convergence Rate of GAUSSIANIZATION: As a contribution to [1], we cover an empirical extension of the theoretical analysis of the convergence behaviour of GAUSSIANIZATION. We investigate the convergence behaviour of GAUSSIANIZATION for both low-dimensional toy datasets and high-dimensional image datasets.

Scaling GAUSSIANIZATION to complex, high-dimensional data remains challenging. The theoretically predicted lower-bound scaling behaviour of $\Omega(D)$ can be recovered for some distributions, while others show more favourable convergence.

Optimal Transport FLOW MATCHING: In parallel to [44], [45], we introduce a novel approach to straightening flow trajectories and reducing training objective stochasticity via mini-batch Optimal Transport. The matching in general works well for high-dimensional data but may incur an error due to the mini-batch approximation [44].

Context-Aware FLOW MATCHING: We apply permutation-invariant neural networks and Optimal Transport FLOW MATCHING for the generation of 3-dimensional point clouds. The random samples are slightly noisier than existing state-of-the-art solutions, but qualitatively look convincing. Reconstructions of samples from the dataset are nearly indistinguishable from the data. This suggests that our architecture is sufficiently adapted to the problem without using tricks like voxelization, which generalise poorly to higher-dimensional data.

In particular, TopK pooling appears to be an attractive alternative to attention-based pooling for data with high set cardinality n due to its low computational complexity of $\mathcal{O}(n+k)$ compared to $\mathcal{O}(n^2)$ for attention or $\mathcal{O}(nm)$ for induced attention with m inducing points [38].

Context-Aware Domain Generalization: As a contribution to [2], we apply a novel approach to Domain Generalization, leveraging permutation-invariant neural networks to extract contextual embeddings from a set of inputs. The contextual information is used to train a robust inference model.

The resulting models can reliably detect failure cases, which allows for model selection at inference time. We formulate three theoretical criteria to evaluate when our approach is effective, which can be verified empirically on real data.

Acknowledgements: The author thanks Felix Draxler, Ullrich Köthe, Jens Müller, Stefan Radev, and Peter Sorrenson for their continued support, mentorship, and insightful feedback. Their expertise and collaborative spirit have been integral to the creation of this thesis.

Furthermore, I extend my sincere gratitude to Felix Draxler, Frederik Kortkamp, and Jens Müller for proofreading and editorial assistance.

Bibliography

[1] F. Draxler, L. Kühmichel, A. Rousselot, J. Müller, C. Schnörr, and U. Köthe, "On the Convergence Rate of GAUSSIANIZATION with Random Rotations." 2023.

[2] J. Müller, L. Kühmichel, M. Rohbeck, S. T. Radev, and U. Köthe, "Towards Context-Aware Domain Generalization: Representing Environments with Permutation-Invariant Networks." 2023.

[3] D. Hendrycks and T. Dietterich, "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations." 2019.

[4] P. W. Koh *et al.*, "WILDS: A Benchmark of in-the-Wild Distribution Shifts." 2021.

[5] B. Bloem-Reddy and Y. W. Teh, "Probabilistic symmetries and invariant neural networks." 2020.

[6] H. Edwards and A. Storkey, "Towards a Neural Statistician." 2017.

[7] K. Muandet, D. Balduzzi, and B. Schölkopf, "Domain Generalization via Invariant Feature Representation." 2013.

[8] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain Generalization: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20, 2022, doi: 10.1109/tpami.2022.3195549.

[9] X. Liu, C. Gong, and Q. Liu, "Flow Straight and Fast: Learning to Generate and Transfer Data with RECTIFIED FLOW." 2022.

[10] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "FLOW MATCHING for Generative Modeling." 2023.

[11] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan, "PointFlow: 3D Point Cloud Generation with CONTINUOUS NORMALIZING FLOWS." 2019.

[12] L. Wu *et al.*, "Fast Point Cloud Generation with Straight Flows." 2022.

[13] "Generative Models." [Online]. Available: https://openai.com/research/generative-models

[14] C. Bunne and M. Cuturi, "Optimal Transport in Learning, Control, and Dynamical Systems," in *ICML'23: Proceedings of the 40th International Conference on Machine Learning*, Honolulu, Hawaii, USA: JMLR.org, 2023. [Online]. Available: http://bunne.ch/ot_tutorial/

[15] P. Sorrenson, F. Draxler, A. Rousselot, S. Hummerich, L. Zimmermann, and U. Köthe, "Lifting Architectural Constraints of Injective Flows." 2023.

[16] F. Draxler, P. Sorrenson, L. Zimmermann, A. Rousselot, and U. Köthe, "Free-form Flows: Make Any Architecture a NORMALIZING FLOW." 2023.

[17] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear Independent Components Estimation." 2015.

[18] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using Real NVP." 2017.

[19] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural Spline Flows." 2019.

[20] S. Chen and R. Gopinath, "GAUSSIANIZATION," in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., 2000.

[21] V. Laparra, G. Camps-Valls, and J. Malo, "Iterative GAUSSIANIZATION: From ICA to Random Rotations," *IEEE Transactions on Neural Networks*, vol. 22, no. 4, pp. 537–549, Apr. 2011, doi: 10.1109/tnn.2011.2106511.

[22] B. Dai and U. Seljak, "Sliced Iterative NORMALIZING FLOWS," in *International Conference on Machine Learning*, 2021.

[23] C. Meng, Y. Song, J. Song, and S. Ermon, "GAUSSIANIZATION Flows." 2020.

[24] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural Ordinary Differential Equations." 2019.

[25] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models." 2018.

[26] M. Hutchinson, "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines," *Communication in Statistics- Simulation and Computation*, vol. 18, pp. 1059–1076, 1989, doi: 10.1080/03610919008812866.

[27] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models." 2020.

[28] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent DIFFUSION MODELS." 2022.

[29] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-Based Generative Modeling through Stochastic Differential Equations." 2021.

[30] T. Salimans and J. Ho, “Progressive Distillation for Fast Sampling of Diffusion Models.” 2022.

[31] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, “Consistency Models.” 2023.

[32] Y. Song and P. Dhariwal, “Improved Techniques for Training Consistency Models.” 2023.

[33] A. Tong *et al.*, “Simulation-free Schrödinger bridges via score and flow matching.” 2023.

[34] A. Davtyan, S. Sameni, and P. Favaro, “Efficient Video Prediction via Sparsely Conditioned Flow Matching.” 2023.

[35] F. Draxler, C. Schnörr, and U. Köthe, “Whitening Convergence Rate of Coupling-based Normalizing Flows.” 2022.

[36] minutephysics and H. Reich, “Simpson's Paradox.” [Online]. Available: https://www.youtube.com/watch?v=ebEkn-BiW5k

[37] C. R. Charig, D. R. Webb, S. R. Payne, and J. E. Wickham, “Comparison of treatment of renal calculi by open surgery, percutaneous nephrolithotomy, and extracorporeal shockwave lithotripsy.,” *BMJ*, vol. 292, no. 6524, pp. 879–882, Mar. 1986, doi: 10.1136/bmj.292.6524.879.

[38] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh, “Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks.” 2019.

[39] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, “Towards Sparse Hierarchical Graph Classifiers.” 2018.

[40] B. Knyazev, G. W. Taylor, and M. R. Amer, “Understanding Attention and Generalization in Graph Neural Networks.” 2019.

[41] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, “Deep Sets.” 2018.

[42] R. Sinkhorn, “A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices,” *The Annals of Mathematical Statistics*, vol. 35, no. 2, pp. 876–879, 1964, doi: 10.1214/aoms/1177703591.

[43] P. Knopp and R. Sinkhorn, “Concerning nonnegative matrices and doubly stochastic matrices.,” *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[44] A. Tong *et al.*, “Improving and generalizing flow-based generative models with minibatch optimal transport.” 2023.

[45] A.-A. Pooladian, H. Ben-Hamu, C. Domingo-Enrich, B. Amos, Y. Lipman, and R. T. Q. Chen, “Multisample FLOW MATCHING: Straightening Flows with Minibatch Couplings.” 2023.

[46] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, no. 1–2, pp. 83–97, 1955, doi: https://doi.org/10.1002/nav.3800020109.

[47] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, “EMNIST: an extension of MNIST to handwritten letters.” 2017.

[48] Z. Wu *et al.*, “3D ShapeNets: A Deep Representation for Volumetric Shapes.” 2015.

[49] L. Kühmichel, “Blender-Plot.” [Online]. Available: https://github.com/LarsKue/blender-plot

[50] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, “Invariant Risk Minimization.” 2020.

[51] H. Fanaee-T, “Bike Sharing Dataset.” [Online]. Available: https://archive.ics.uci.edu/dataset/275

[52] A. F. Agarap, “Deep Learning using Rectified Linear Units (ReLU).” 2019.

[53] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, “Self-Normalizing Neural Networks.” 2017.

[54] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer Normalization.” 2016.

[55] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” 2015.

[56] L. H. Zhang, V. Tozzo, J. M. Higgins, and R. Ranganath, “Set Norm and Equivariant Skip Connections: Putting the Deep in Deep Sets.” 2022.

[57] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization.” 2019.

[58] L. N. Smith and N. Topin, “Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.” 2018.

[59] T. B. Brown *et al.*, “Language Models are Few-Shot Learners.” 2020.

[60] W. Falcon and The PyTorch Lightning team, “PyTorch Lightning.” [Online]. Available: https://github.com/Lightning-AI/lightning

A Appendix

A.1 Loss Functions

1. Balanced Cross Entropy:

$$H(x, y) = -\sum_{c=1}^{C} \log \operatorname{softmax}(x_c) y_c \tag{47}$$

where the softmax is drawn over all classes c.

2. Mean-Squared Error:

$$\operatorname{MSE}(x, y) = \|x - y\|_2^2 \tag{48}$$

3. Maximum-Mean-Discrepancy:

$$\operatorname{MMD}(X, Y) = \operatorname{kernel}(X, X) + \operatorname{kernel}(Y, Y) - 2 \cdot \operatorname{kernel}(X, Y) \tag{49}$$

where we choose to use the Gaussian kernel

$$\operatorname{kernel}(X, Y) = \exp\left(-\frac{\|X - Y\|_2^2}{2\sigma}\right) \tag{50}$$

with $\sigma = 1$. Multiple non-unit σ can be used and averaged over to improve the training signal. For all experiments that use MMD, we choose σ roughly between 10^{-6} and 10^{6}: `sigma = torch.logspace(-20, 20, base=2, steps=41)`.

4. Chamfer Distance:

$$\operatorname{CD}(X, Y) = \sum_{x \in X} \min_{y \in Y} \|x - y\|_2^2 + \sum_{y \in Y} \min_{x \in X} \|x - y\|_2^2 \tag{51}$$

A.2 Activation Functions

ReLU [52]:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{52}$$

SELU [53]:

$$\mathrm{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases} \tag{53}$$

softmax:

$$\mathrm{softmax}(x) = \frac{\exp(x_i)}{\sum_i \exp(x_i)} \tag{54}$$

A.3 Algorithms

```python
import torch
from torch import Tensor

def sinkhorn(x: Tensor, y: Tensor, sigma: float, steps: int) -> Tensor:
    log_cost = -0.5 * torch.cdist(x, y) ** 2 / sigma

    for _ in range(steps):
        log_cost = torch.log_softmax(log_cost, dim=0)
        log_cost = torch.log_softmax(log_cost, dim=1)

    return log_cost.exp()
```

Listing 2: Sinkhorn-Knopp Optimal Transport algorithm for Euclidean transport cost.

To match a batch of sets as shown in Listing 4, we can simply call the sinkhorn algorithm for every item in the batch:

```python
import torch
from torch import Tensor

def set_match(x: Tensor, y: Tensor, sigma: float, steps: int):
    batch_size, set_size, *_ = x.shape
    for i in range(batch_size):
        log_pi = sinkhorn(x[i], y[i], sigma, steps)
        permutation = torch.multinomial(log_pi.exp(), 1)
        permutation = permutation.squeeze(1)

        y[i] = y[i, permutation]

    return x, y
```

Listing 3: Sinkhorn-Knopp Optimal Transport algorithm for a batch of sets.

Future implementations may use `torch.vmap` instead, assuming that data-dependent control flow is implemented, which is required for the convergence check. See this issue. The current workaround is to apply the matching within the dataset instead.

A.4 Experiment Details

Details for experiments that are contributions to [1] and [2] can be found in the respective papers.

A.4.1 Context-Aware Flow Matching

Architecture: For the set-encoder as shown in Figure 21, we choose to interleave fully connected residual layers with TopK pooling layers [39], [40], halving the set size with each pooling. We chose TopK pooling due to its ability to retain a non-unit part of the set (unlike mean pooling), while also being much faster than attention pooling. Each TopK pooling is preceded by a non-activated residual linear layer to allow the network to map values to an arbitrary range, followed by a LayerNorm [54] to return values to a normalised range.

The final ninth pooling is a mean pooling, which we use due to its increased expressive power over the TopK pooling.

We choose SELU [53] as the activation function for the network for its normalising effect on very deep newtorks.

For the flow, we choose a similar architecture, successively applying residual linear blocks and SELU activations. The flow contains no pooling layers since it acts independently on each item in the set.

We experimented with applying different normalisation layers, such as BatchNorm [55] or SetNorm [56], for both the fully-connected part of the set-encoder, as well as the flow. However, we found these to yield no benefit. Training was often less stable using these layers or could not converge at all.

Initialisation: We initialise the set-encoder as a random projection and the flow as a random drift by zeroing the bias of each linear layer and computing its weight matrix as

$$W_{ij} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{n_{\text{out}}^2}\right) \tag{55}$$

where n_{out} is the number of output neurons. This initialization is much closer to zero than the default and thus ensures stability at the start of training.

Memory Tricks: There are several tricks we can apply to keep GPU memory within a feasible range. Firstly, we reduce the effective batch size for the flow by taking a subset from the input set of points. The effective batch size for HParams as shown in Table 4 is thus $2 * 256 * 256 = 131072$ points, which is sufficiently large for a good training signal.

Secondly, we may trade compute for memory by accumulating the gradient over multiple batches in order to achieve a larger effective batch size with no additional memory cost. This may be necessary if the set-encoder requires even larger batch sizes that are prohibitively expensive for the flow.

Lastly, we use activation checkpointing with 16 total segments for both the set-encoder and the flow. This also trades compute for memory; the memory gain is roughly on the order of 8GiB for the hyperparameters as shown in Table 4. Note that typically, it will be faster to use checkpointing instead of reducing the batch size and accumulating multiple batches, since here we only pay the extra compute cost in the gradient backpropagation step.

Furthermore, we can significantly speed up the training procedure by offloading the Optimal Transport matching to a worker process. As such, although technically valid, the training algorithm as shown in Listing 4 should not be used in practice. Refer to the git repository as linked in Appendix A.5.2 instead for an efficient implementation.

Furthermore, since execution speed is typically bound by the Optimal Transport matching, none of the memory saving operations described above truly trade execution speed, instead only serving to save memory at the cost of increased energy usage.

Pre-Training: We pre-train a base model with the following hyperparameters:

HParam	Value	Details
`batch_size`	256	-
`subset_size`	256	Governs the effective batch size for the flow
`embeddings`	768	Dimension of the set-encoder latent space
`accumulate_batches`	2	Increases the effective batch size
`max_epochs`	10^4	-
`gamma`	0.5	Governs the strength of MMD to MSE respectively
`optimizer`	AdamW	See [57]
`lr_scheduler`	OneCycleLR	See [58]
`max_lr`	10^{-4}	The peak learning rate after 30% of epochs
`div_factor`	10	Controls the initial learning rate
`final_div_factor`	10^6	Controls the final learning rate
`gradient_clip`	1	Clips total gradient norm at this value
`weight_decay`	0.01	Decoupled from the learning rate, see [57]
`epsilon`	0.05	Optimal Transport entropy regularization

Table 4: Hyperparameters used for pre-training a base model on ModelNet10 point clouds.

Fine-Tuning: Throughout hyperparameter optimisation, we experiment with fine-tuning models by initialising a new training run with weights from the latest checkpoint. We typically increase the `batch_size`, `subset_size`, `accumulate_batches`, and `gradient_clip` parameters for a finer estimation of the gradient in this step, while reducing the `max_lr` and `epsilon`.

Supplement:

```python
import torch
from torch import Tensor

from losses import mmd_loss, mse_loss
from networks import encoder, flow
from optimal_transport import set_match

def loss(Sn: Tensor, gamma: float) -> Tensor:
    t = torch.rand(Sn.shape[0], Sn.shape[1], 1)

    c = encoder(Sn)
    encoder_loss = mmd_loss(c, torch.randn_like(c))

    # permute within each set
    x0, x1 = set_match(torch.randn_like(Sn), Sn)

    xt = t * x1 + (1 - t) * x0

    predicted_velocity = flow(xt, t, c)
    target_velocity = x1 - x0
            flow_loss    =   mse_loss(predicted_velocity,
target_velocity)

    return gamma * encoder_loss + (1 - gamma) * flow_loss
```

Listing 4: Simplified example code for the loss as shown in (40).

A.5 Tools

This thesis was written with the help of large AI language models, such as [59], to enhance linguistic clarity and conciseness, as well as to review and adjust text formulation. The primary content, ideas, and research presented in this thesis are the original work of the author. AI language models were used solely as a supportive tool to improve the quality of the written presentation. Any text proposed by such models was carefully reviewed and, where necessary, adjusted by the author prior to inclusion in the thesis.

Furthermore, in order to perform the experiments described in Section 4, we make extensive use of the Python ecosystem and its rich set of deep learning software packages, most notably

Software	Version
Python	3.11
PyTorch	2.0
Lightning	2.0
Lightning-Trainable	0.4.0-rc2

A full list of dependencies can be found on each respective git repository.

A.5.1 Lightning Trainable

As part of the experiments highlighted in this thesis, we develop a high-level prototyping and experiment library based on PyTorch Lightning [60], dubbed Lightning-Trainable. With this library, we encourage models to be entirely reconstructable from a set of hyperparameters, which helps speed up experimentation while allowing us to reliably reproduce experiments even without knowledge of the parameters used for earlier training.

It also lowers the entry barrier to using PyTorch Lightning for scalable deep learning by providing a set of useful and convenient configuration options as well as a fully pre-configured trainable module.

Lightning-Trainable was used in the creation of both papers, [1] and [2]. We encourage you to check out the library at https://github.com/LarsKue/lightning-trainable/.

A.5.2 Software Repositories

Our experiments and related software are publicly available on GitHub:

https://github.com/vislearn/Gaussianization-Bound/

https://github.com/LarsKue/context-aware-flow-matching/

https://github.com/LarsKue/lightning-trainable/

https://github.com/LarsKue/blender-plot/

and soon to be public:

https://github.com/XarwinM/adaptive_dg/