Consistency Models: Core Principles and Comparative Analysis with Diffusion Models, VAEs, and Normalizing Flows

Gia Ancone[a,b] **and Scott Le Roux**[b]

[a]Stanford University; [b]Oxford University

This manuscript was compiled on June 4, 2026

This report provides an overview of consistency models, a new class of generative models that enforce a self-consistency constraint along Probability Flow ODE (PF-ODE) trajectories in order to achieve high fidelity sampling in one step. First, the report reviews diffusion models and their PF-ODE formulation, as they provide the foundation for consistency models. Then, consistency functions, boundary conditions, and training procedures via consistency distillation are detailed. After this, consistency models are compared conceptually and mechanically to other popular generative models studied in this tutorial, including diffusion models, VAEs, and normalizing flows. To provide some supporting evidence behind our claims, we also implement a 2D mixture-of-Gaussians generation experiment. We compare learned trajectories and sample quality for a score-based diffusion model and a distilled consistency model, and show that the latter recovers the target distribution with roughly 100× fewer network evaluations and over 60× lower sampling time. We conclude by discussing current limitations and next steps for development of consistency models.

Consistency Model | Diffusion Model | Generative Model | Sampling

Recently, diffusion models (Ho et al. (2020)) have been the driving force behind generative modeling, and allowed us to achieve state-of-the-art performance in image, audio, and video synthesis. The strategy is as follows: gradually adding noise to the data, and then learning to reverse this corruption process via iterative denoising steps. This modeling procedure is incredibly effective for producing high quality results, and offers opportunities to execute quality image inpainting, super resolution, and general inverse problem solving. However, thousands of neural network inferences are required per sample, which makes them orders of magnitude slower than other, single-step generation models like GANs (Goodfellow et al. (2014)), VAEs (Kingma and Welling (2022)), and normalizing flows (Rezende and Mohamed (2016)). While these models are much faster, they lack the robustness of diffusion models, and often fail to produce as physically realistic results. Hence, there is an obvious need for a generative model that embodies both the generation quality and flexibility of diffusion models with the rapidity of these single-step generation models.

Consistency models have been designed to address this need (Song et al. (2023)). Consistency models are a new family of models that "generate high quality samples by directly mapping noise to data". With these models, the goal is to learn a consistency function that maps any point along a Probability Flow ODE back to its corresponding data point. In enforcing a self-consistency constraint, which requires that every noisy sample variant maps back to the original clean image, consistency models are able to generate high quality images in a single forward pass, with optional multistep refinement.

This report offers an explanation of the core mechanics of consistency models. As this final project is a cumulative demonstration of what I have learned in my Oxford University Unsupervised & Probabilistic Machine Learning: Focus on Generative Models tutorial, special attention will be given to addressing the similarities and differences in process and performance of consistency models, classic diffusion models, VAEs, and normalizing flow models. This report focuses on the central mathematical structure and intuition behind model function to help foster an understanding of how consistency models operate and why they are an important development in generative modeling. This report accomplishes the following:

1. Reviews diffusion models and their Probability Flow ODE formulation as the foundation on which consistency models are built.
2. Defines consistency models, including consistency functions, boundary conditions, and skip-connection parameterization.
3. Describes sampling procedures for consistency models, covering both single-step generation and optional multistep refinement.
4. Explains Consistency Distillation (and briefly Consistency Training) as training methods for learning the consistency mapping from a diffusion model 'teacher' or from scratch.
5. Compares the mechanics, assumptions, and failure modes of consistency models with diffusion models, VAEs, and normalizing flows, and illustrates these differences with a 2D mixture-of-Gaussians implementation.

1. Background: Diffusion Models

As mentioned previously, diffusion models (Ho et al. (2020)) are generative models that repeatedly add Gaussian noise to data until it matches a tractable noise distribution (Gaussian usually). Because this process is analytically defined, it provides supervision for training a model to approximate the reverse process, denoising, where sampled noise is transformed back into realistic data, reminiscent of the original data distribution.

Given a data distribution $p_{data}(x)$, diffusion models define a forward stochastic differential equation (SDE). This SDE gradually perturbs the data into noise via Gaussian corruption. This SDE is of the form

$$dx_t = \mu(x_t, t)dt + \sigma(t)\, dw_t \qquad [1]$$

[2]To whom correspondence should be addressed. E-mail: gancone@stanford.edu

where $t \in [0, T], x_0 \sim p_{data}$, and $\sigma(t)$ controls the rate at which noise is injected. Most often, the SDE is chosen such that the terminal noise distribution $p_T(x)$ is close to an isotropic Gaussian, which makes sampling tractable.

Whenever the forward SDE admits a well-defined probability density p_t, we can derive a corresponding Probability Flow ODE whose time-marginal distributions match those of the SDE. The Probability Flow ODE is given by

$$\frac{dx_t}{dt} = \mu(x_t, t) - \frac{1}{2}\sigma(t)^2 \nabla \log p_t(x_t) \quad [2]$$

There are two ways to describe this. You can envision many random particles following the SDE in their own way, and their empirical histogram at time t is the $p_t(x)$. Or, you can take the fluid view (ODE + continuity equation). You can imagine a continuous fluid of probability mass flowing in space, driven by a velocity field $v(x, t)$. The density will satisfy the following continuity equation:

$$\frac{\partial p_t(x)}{\partial t} = -\nabla_x \cdot \big(p_t(x) v(x, t)\big) \quad [3]$$

If we know $v(x, t)$, then we can think of each point x_t moving deterministically, with $\frac{dx_t}{dt} = v(x_t, t)$. The PF-ODE is exactly this deterministic fluid/velocity description, with

$$v(x, t) = \mu(x, t) - \frac{1}{2}\sigma(t)^2 \nabla_x \log p_t(x).$$

This is deterministic, so there is no randomness, no Brownian motion, and no sampling of noise during the trajectory. And yet, the distribution of x_t at each time t exactly matches the SDE's distribution. How is this possible? Because the PF-ODE moves points in probability space in exactly the way that the SDE does on average. With SDE, if you start with an x_0 and run SDE many times, each run offers a different x_t and when taken together, these random outcomes form a distribution $p_t(x)$. As t increases, this distribution will more closely resemble a Gaussian.

However, if we instead run the PF-ODE backwards in time, starting from a point drawn from the terminal distribution, we will still have a single-valued trajectory because there is no randomness. If we do this for all possible starting points at time T, the collection of resulting points at earlier times t will give us the same $p_t(x)$ distribution as the SDE.

Now, in the above PF-ODE, the $\mu(x_t, t)$ term is the drift term from the SDE and $\sigma(t)$ controls how much noise is added in the SDE. $\nabla \log p_t(x_t)$ is the score function, meaning the gradient of the log-density of the noised data at time t. This term tells us which direction increases the likelihood of the noisy data. This means that it points toward higher-probability regions.

Now, we do not know the actual score function $\nabla \log p_t(x_t)$, because we don't know the analytic form of the noised distribution. The original data distribution $p_0(x)$ is unknown, so the noised version, $p_t(x)$ is also unknown analytically. So, diffusion models train a neural network,

$$s_\phi(x, t)$$

to approximate the score function using score matching. This is why diffusion models are also known as score-based generative models. Replacing the true score with the estimation given by s_ϕ gives us an empirical PF-ODE that can be solved backward from pure noise at time $t = T$ to a sample resembling the data at time 0.

The PF-ODE, thus, enables us to move a point x_t a tiny amount from time t to time $t - \Delta t$ in the direction that makes the sample a bit less noisy using the score network. Because we cannot jump directly from $t = T$ to $t = 0$, we must execute this process gradually, across many small steps. At each discrete timestep, we solve the ODE numerically by updating the sample,

$$x_{t_k} \leftarrow x_{t_{k+1}} + \text{update that uses } s_\phi(x_t, t).$$

The model evaluates the score network and moves the sample slightly toward a more "denoised" direction during each step, repeated N times for each of the N discrete timesteps.

In explaining this process, it becomes obvious how computationally intensive this generation process is. Every step means running the score neural network inference, conducting intensive linear algebra, and writing the updated sample to memory. The more steps you need, the slower the sampling, so time required increases with quality of generation. Although there are some existing means of acceleration, such as improved numerical solvers and distillation methods, most still require many steps or depend on computationally expensive data generation pipelines. Consistency models were created to address this problem.

2. Consistency Models

A. Consistency Functions. Consistency models are a new type of model created by Song et al. (2023) that "support single-step generation" at their core, while still "allowing iterative generation for trade-offs between sample quality and compute". To understand why consistency models work, we must make these two observations:

1. A Probability Flow ODE (PF-ODE) defines a smooth trajectory that transforms a data point x_0 into a noise sample x_T.

2. Every point on this trajectory belongs to the same underlying sample, as it is just the same image at different noise levels.

Let's define a sequence of noise levels

$$0 < t_1 < t_2 < ... < t_K = T$$

and the corresponding states as

$$x_{t_1} \rightarrow x_{t_2} \rightarrow ... \rightarrow x_{t_K}$$

We choose t_1 as the cleanest usable state and $t_K = T$ is the noisiest state. Then, the consistency function is

$$f(x_t, t) \rightarrow x_{t_1} \quad [4]$$

This means that you can give the function any noisy version of the sample, at any point on the trajectory and the function will output the denoised version at time t_1 (nearly clean image). No matter which version of a particular image you give the model, it is required to return the same "starting point" of that trajectory. See Figure 1.

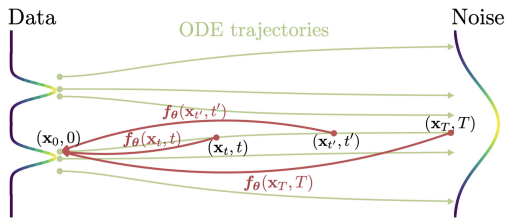

Fig. 1. "Consistency models are trained to map points on any trajectory of the PF ODE to the trajectory's origin" (Song et al. (2023)). This figure demonstrates how consistency models learn to map any point along a PF-ODE trajectory back to the same underlying data sample. In the diagram, points such as x_T, x_t, and $x_{t'}$ lie on the same PF-ODE path but represent different noise levels. The model $f_\theta(x, t)$ is trained such that all of them are sent to (approximately) the same reconstruction $x_\epsilon \approx x_0$. This enforced self-consistency ensures that after training, a single evaluation of the model can recover the clean data from any noisy input.

B. Boundary Conditions. Actually, diffusion models can never go all the way to time 0 as this would create instability. Instead, we define a small positive time ϵ that is "almost clean". Hence, the consistency function must satisfy a boundary condition at time $t = \epsilon$:

$$f(x_\epsilon, \epsilon) = x_\epsilon \quad [5]$$

If the input is already almost clean at time ϵ, then the network should leave it unchanged, essentially acting as an identity function at $t = \epsilon$.

This boundary condition is crucial. Without it, the trivial solution $f(x, t) = 0$ could satisfy the consistency constraint for all pairs of points on a trajectory. Every input, no matter the content or noise level would be mapped to the same constant output. This collapse means that the generative model would not preserve any information about the underlying data, since it does not differentiate between different samples or trajectories. Hence, this boundary condition forces the network to reproduce x_ϵ exactly, which anchors the model and rules out constant or otherwise degenerate solutions. Now, how do we force a neural network to always respect this condition?

There are two methods for designing the model to satisfy the boundary condition, detailed in Song et al. (2023).

Option 1: Hard-Coded Identity at $t = \epsilon$

$$f_\theta(x, t) = \begin{cases} x, & t = \epsilon, \\ F_\theta(x, t), & t > \epsilon. \end{cases} \quad [6]$$

This works, but is not differentiable at $t = \epsilon$, which can pose challenges during continuous-time training.

Option 2: Skip-connection Parametrization

$$f_\theta(x, t) = c_{skip}(t)x + c_{out}(t)F_\theta(x, t) \quad [7]$$

with the constraints

$$c_{skip}(\epsilon) = 1, \qquad c_{out}(\epsilon) = 0. \quad [8]$$

Here, $F_\theta(x, t)$ is a typical neural network, while $c_{skip}(t)$ and $c_{out}(t)$ are time-dependent scalar functions. This form means that some of the input is directly skipped, while we add a learned connection F_θ scaled by a weight. This is like skip-connections in a residual network.

With the above constraints, we are guaranteed that at $t = \epsilon$, the model outputs exactly x, for any parameters θ. Then, for $t > \epsilon$, the model smoothly shifts toward a learned denoiser. For larger noise levels, c_{skip} smoothly decreases and the model relies less on copying the input. c_{out} smoothly increases and the model will rely more on the learned denoiser F_θ. A simple example of such scaling functions are

$$c_{\text{skip}}(t) = 1 - \frac{t - \epsilon}{T - \epsilon}, \qquad c_{\text{out}}(t) = \frac{t - \epsilon}{T - \epsilon}, \quad [9]$$

which linearly fade the skip connection out and the denoiser in as t grows.

This is useful because you can reuse the same main network architecture and can just change the output head using c_{skip} and c_{out}. Thus, we do not need to train the "identity at ϵ" behavior, which stabilizes training. Optimization will be easier, because the model is not forced to learn a discontinuous jump from identity to denoiser. The scaling functions ensure a smooth morphing, which improves gradient flow.

C. Sampling with Consistency Models. Sampling begins by drawing an initial latent vector from the terminal noise distribution of the Probability Flow ODE, typically $x_T \sim \mathcal{N}(0, T^2 I)$. The consistency model is trained to map any point on a PF-ODE trajectory back to its corresponding near-clean endpoint x_ϵ. Applying the learned consistency function once, we compute $x_\epsilon = f_\theta(x_T, T)$. This x_ϵ should be interpreted as $x_\epsilon \approx x_0$, the generated image. This is one-step sampling, incredibly fast compared to diffusion models that need $10 - 1000$ rounds of neural network inference.

There are obvious computational advantages to conducting one-step sampling. However, consistency models also allow for an optional multistep refinement procedure. This procedure is analogous to iterative diffusion sampling, but still much more efficient.

In practice, the intermediate timesteps $\tau_1, \tau_2, \ldots \tau_{N-1}$ are typically selected using a noise-level schedule. They are often spaced more densely near ϵ, where smaller refinements matter most, and more sparsely at larger noise levels. Unlike diffusion samplers that follow a predefined β_t schedule (Ho et al., 2020), the timesteps τ_n used in consistency sampling are not aligned with the forward diffusion noise schedule. Instead, they are chosen directly in continuous PF-ODE time according to a noise-level schedule or tuning heuristic (Song et al., 2023).

Algorithm 1 Multistep Consistency Sampling. Starting from a noisy input $\hat{x}_T$, the algorithm alternates between injecting the correct noise level for time τ_n, and applying the consistency model f_θ. Repeating this process produces a more refined sample.

Require: Consistency model $f_\theta(\cdot, \cdot)$; time points $\tau_1 > \tau_2 > \cdots > \tau_{N-1} > \epsilon$; initial noise $\hat{x}_T$

$x \leftarrow f_\theta(\hat{x}_T, T)$
for $n = 1$ **to** $N - 1$ **do**
  Sample $z \sim \mathcal{N}(0, I)$
  $\hat{x}_{\tau_n} \leftarrow x + \sqrt{\tau_n^2 - \epsilon^2}\, z$
  $x \leftarrow f_\theta(\hat{x}_{\tau_n}, \tau_n)$
return x

First, we have an initial denoising step. $\hat{x}_T \sim \mathcal{N}(0, T^2 I)$ is sampled from the terminal PF-ODE noise distribution. The model maps directly to an estimate of x_ϵ, which is the clean endpoint. Subsequent iterations simulate the PF-ODE

marginal at intermediate times by adding appropriately scaled Gaussian noise.

We sample z from a simple distribution, typically $z \sim \mathcal{N}(0, I)$. To understand the update

$$\hat{x}_{\tau_n} \leftarrow x + \sqrt{\tau_n^2 - \epsilon^2}\, z, \tag{10}$$

it helps to first consider the idealized PF-ODE form. Under the PF-ODE, a point at time t satisfies the simple analytic expression $x_t = x_0 + tz$, which means that the marginal distribution is $x_t \sim \mathcal{N}(x_0, t^2 I)$. If we had actual access to x_0, generating a noisy point at any time t would just require adding Gaussian noise of standard deviation t.

However, the consistency model is incapable of directly outputting x_0. Its first prediction is x_ϵ, the slightly noisier version of the cleanest sample. Hence, the marginal form is

$$x_t = x_\epsilon + (t - \epsilon)z,$$

But, the PF-ODE variance is t^2, not $(t-\epsilon)^2$. Expansion gives us

$$x_{\tau_n} = x_\epsilon + \tau_n z = (x_\epsilon + \epsilon z) + \underbrace{(\tau_n z - \epsilon z)}_{\text{extra noise}}.$$

Because the total variance at time t is t^2, the additional variance needed to move from time ϵ to τ_n is $\tau_n^2 - \epsilon^2$. Hence, the incremental noise must have magnitude $\sqrt{\tau_n^2 - \epsilon^2}$, guaranteeing that $\hat{x}_{\tau_n}$ is distributed exactly according to the PF-ODE marginal at time τ_n.

After this, the consistency model removes that noise at the corresponding time level. This is the step $x \leftarrow f_\theta(\hat{x}_{\tau_n}, \tau_n)$. The consistency function is applied at the new noise level. If the initial estimate x_ϵ is imperfect, the refinement steps help correct it, as early (large t) denoising will remove coarse noise, while later steps refine smaller details. As τ_n decreases, the noise becomes smaller and the model refines smaller details. Even 1-4 revisions are useful for high-quality results.

D. Training Consistency Models via Distillation. The goal is to train a consistency model $f_\theta(x_t, t)$ such that it can map any noisy point x_t along a diffusion ODE trajectory back to the corresponding clean endpoint x_ϵ, meaning that

$$f_\theta(x_t, t) \approx f_\theta(x_{t'}, t') \approx x_\epsilon$$

for any two times t, t' on the same trajectory. Because the true PF-ODE is unknown, we instead use a pretrained diffusion model as a teacher and generate adjacent points on the empirical PF-ODE using a single ODE step. This procedure is known as consistency distillation because we are distilling the diffusion model's deterministic PF-ODE denoising mapping, namely the multistep trajectory that maps x_T to x_ϵ, into a single step function f_θ.

The trajectory x_t satisfying the "true" PF-ODE

$$\frac{dx_t}{dt} = -t \nabla \log p_t(x_t) \tag{11}$$

is unknown, because the score $\nabla \log p_t$ is unknown. It is the exact score of the unknown intermediate noisy distributions p_t. However, we can approximate it using a pretrained diffusion model that includes a score network $s_\phi(x, t)$. This is the reason for distillation: we use the diffusion model as a teacher that can approximate PF-ODE trajectories, and we train the consistency model to collapse those multistep trajectories into a single-step mapping. Hence, $s_\phi(x, t) \approx \nabla \log p_t(x)$. We replace the unknown score with the learned one, the empirical PF-ODE:

$$\frac{dx_t}{dt} = -t\, s_\phi(x_t, t) \tag{12}$$

Reminder, we care about this ODE because if we solve this ODE backward (from large to small noise), we move deterministically from noise to data. To train this return-to-origin mapping, we need examples of adjacent points on the same PF-ODE trajectory. Hence, we use the diffusion model to generate PF-ODE trajectories, employing a diffusion model like a teacher. We cannot sample the exact PF-ODE trajectory, so we use the empirical PF-ODE to approximate it.

For a clean data sample $x \sim p_{data}$, we do the following:

1. Sample a noisy version at time
   $t_{n+1} : x_{t_{n+1}} \sim \mathcal{N}(x, t_{n+1}^2 I)$.
2. Take one backward ODE step from $t_{n+1} \to t_n$:

$$\hat{x}^\phi_{t_n} = x_{t_{n+1}} + (t_n - t_{n+1})\, \Phi\big(x_{t_{n+1}},\, t_{n+1};\, \phi\big) \tag{13}$$

   This equation defines how to compute an ODE-based estimate of a slightly cleaner PF-ODE point x_{t_n} using a single numerical solver step starting from the noisier point $x_{t_{n+1}}$. The quantity $\hat{x}^\phi_{t_n}$ is therefore a predicted version of the next (slightly cleaner) point along the trajectory at time t_n. The superscript ϕ indicates that the estimate depends on the pretrained diffusion model's score network.

   $\Phi(x, t; \phi)$ denotes the update rule of a numerical ODE solver applied to the empirical Probability Flow ODE,

$$\frac{dx_t}{dt} = -t\, s_\phi(x_t, t).$$

   This matches the standard ODE update rule

$$x(t + \Delta t) \approx x(t) + \Delta t\, f(x(t), t),$$

   except that here the update is taken backward in time (i.e., $\Delta t = t_n - t_{n+1} < 0$) because sampling integrates the PF-ODE from noise toward data.

Now, we have $x_{t_{n+1}}$ (the noisier sample from the SDE) and $\hat{x}^\phi_{t_n}$ (the less noisy ODE-predicted sample). This pair is used to enforce self-consistency:

$$f_\theta(x_{t_{n+1}}, t_{n+1}) \approx f_{\theta'}(\hat{x}^\phi_{t_n}, t_n) \approx x_\epsilon$$

To enforce this requirement, we train the consistency model using the difference between these two predictions. Consistency Distillation loss:

$$L_{\text{CD}} = \mathbb{E}\big[\lambda(t_n)\; d\big(f_\theta(x_{t_{n+1}},\, t_{n+1}),\; f_{\theta'}(\hat{x}_{t_n},\, t_n)\big)\big] \tag{14}$$

In this equation, $d =$ the distance metric, θ' is the EMA target network, and θ is the online network. $\lambda(t_n)$ is a noise level weighting function that balances the relative contribution of different timesteps to the loss. In practice, it is chosen to down-weight very small t values (where gradients are unstable) and up-weight noisier states where the teacher signal is stronger.

The online network θ is the network we are actively training. Its weights are updated every training step using gradient descent. It receives the gradients of the Consistency Distillation

loss. It is the freshest, most rapidly changing version of the model, $\theta \leftarrow \theta - \eta \nabla_\theta L_{\text{CD}}$.

The target network θ' is a smoothed, and more slowly updated version of the model. It does not receive gradients directly, instead its parameters are updated by an exponential moving average of the online network, $\theta' \leftarrow \mu\theta' + (1-\mu)\theta$, where $\mu \in [0, 1)$ is close to 1. This network is much more stable, as it filters out the rapid fluctuations introduced by stochastic gradient descent. Each update of θ depends on a different minibatch, a randomly sampled timestep, and freshly sampled noise. All of these cause high-variance gradient steps. These rapid shifts make the online model a fast-moving target, which can destabilize training. Hence, using θ' as the teacher prevents this: the student learns from a consistently behaving model instead of one that changes at every iteration. At inference time, the final consistency model is the EMA network $f_{\theta'}$, not the online network f_θ. This is because $f_{\theta'}$ averages many historical versions of the model, so it produces more stable and higher-quality predictions. The online network will exist only for training, whereas the EMA network is the one deployed for sampling. Throughout this paper, we use $f_\theta(x, t)$ to refer to the consistency model network for simplicity, but we actually mean $f_{\theta'}(x, t)$.

Consistency models rely on enforcing self-consistency, $f_\theta(x_{t_{n+1}}, t_{n+1}) \approx f_{\theta'}(\hat{x}^\phi_{t_n}, t_n)$. If both sides used the rapidly-changing online network θ, both predictions would shift too quickly, the loss would become unstable, and thus training would consequently collapse or oscillate. Instead, the online network learns to match the more stable teacher, θ'.

The original Song et al. (2023) paper actually presents two training methods, Consistency Distillation (CD) and Consistency Training (CT). Consistency distillation uses a pretrained diffusion model to provide approximate PF-ODE trajectories. This is the method discussed above. However, Consistency Training (CT) trains a model from scratch without a diffusion model teacher. We will not go into detail about the mechanics of Consistency Training in this report.

Song et al. (2023) provides a theoretical guarantee that if the distillation loss reaches 0 and the ODE solver step size is sufficiently small, then the learned consistency model approximates the true PF-ODE solution with error on the order of the solver's accuracy. This means that a more accurate ODE solver (Heun vs. Euler) will yield a better consistency model. It also means that as the number of discretization steps increases, the learned mapping converges to the empirical PF-ODE flow defined by the pretrained diffusion model, not the unknown true PF-ODE.

Consistency distillation takes the multistep deterministic PF-ODE sampling procedure of a diffusion model and trains a new model f_θ to match its effect in a single evaluation. It does this by enforcing invariance along PF-ODE trajectories, using adjacent ODE samples produced by the pretrained diffusion model as supervision. With the EMA student-teacher paradigm, we are able to ensure stability and give f_θ the ability to learn the underlying flow without collapse. After training, a consistency model can generate images with one-step sampling, while still supporting multistep refinement if called for.

3. Consistency Models vs. Other Generative Models

A. Consistency vs. Diffusion Models. Consistency models are directly derived from the Probability Flow ODE, and employ a diffusion model as a teacher during distillation (detailed above). While both models share the continuous-time generative process, the two families differ in how samples are produced and what the model learns.

1. **Iterative vs. One-Step Sampling**

   In diffusion models, the PF-ODE, $\frac{dx_t}{dt} = -ts_\phi(x_t, t)$ is solved backward from $t = T$ to $t = \epsilon$ using an iterative evaluation process that can take anywhere from 10 to 1000 steps.

   In contrast, consistency models train a function $f_\theta(x_t, t) \approx x_\epsilon$ that ensures that all noisy states on a single PF-ODE trajectory map to one clean origin. This means that sampling reduces to a single forward pass, $x_0 \approx f_\theta(x_T, T)$. Obviously, there is opportunity for multistep refinement, but this is 1. not required, and 2. still very few steps compared to classic diffusion. See Figure 2.

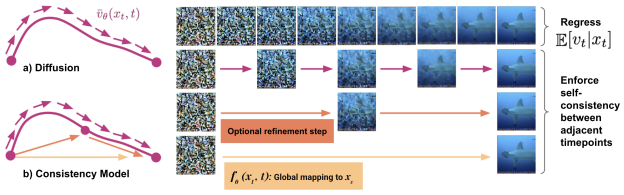

Fig. 2. Comparison of diffusion sampling and consistency model sampling. (**a**) Diffusion models predict a local score or velocity field and must follow the Probability Flow ODE through many small solver steps to denoise an image. (**b**) Consistency models instead learn a global mapping $f_\theta(x_t, t)$ that collapses any noisy state on the PF-ODE trajectory directly to the near-clean point x_ϵ, with optional refinement steps. Adapted from Frans et al. (2025)

.

2. **What the Network Learns**

   The diffusion model learns the score function $s_\phi(x, t) \approx \nabla_x \log p_t(x)$, which is a local vector field representing the score at every time level.

   A consistency model instead learns a global denoising map $f_\theta(x_t, t)$ that collapses an entire PF-ODE trajectory into a single clean point x_ϵ. The entire ODE solution is closed form. With the Consistency Distillation method, you do need the diffusion model teacher's score function $s_\phi(x, t)$ but only to generate training pairs $(x_{t_{n+1}}, \hat{x}_{t_n})$. However, the consistency model itself never learns the score function, it only learns to satisfy $f_\theta(x_{t_{n+1}}, t_{n+1}) \approx f_{\theta'}(\hat{x}_{t_n}, t_n)$. The score is used to compute $\hat{x}_{t_n}$ by the PF-ODE update $\hat{x}_{t_n} = x_{t_{n+1}} + (t_n - t_{n+1})\Phi(x_{t_{n+1}}, t_{n+1}; \phi)$. However, once training is done, the score function is not used again, unlike in diffusion models. Even in consistency training (CT) methods, where there is no diffusion model and the consistency model is trained from scratch, no score function is used.

B. Consistency vs. VAEs. VAEs and consistency models are both single-step generators. However, their assumptions and objectives differ.

1. **Representation and Generative Process**

   A VAE uses an explicit latent variable $z \sim p(z)$, and learns both an encoder $q_\phi(z|x)$ and decoder $p_\theta(x|z)$. In a consistency model, there are no explicit latent variables

and sampling is $x_0 = f_\theta(x_T, T)$, $x_T \sim \mathcal{N}(0, T^2 I)$. See Figures 3, 4.

2. **Objective Function**

   For VAEs, the goal is ELBO maximization, which essentially means optimizing a lower bound on the marginal log-likelihood by jointly encouraging 1. accurate reconstruction of the data, and 2. a latent distribution $q_\phi(z|x)$ that stays close to the prior $p(z)$.

   Consistency models do not use likelihoods or ELBO, they use a consistency loss between neighboring timepoints, L_{CD} in Eq. 14, which enforces self-consistency along PF-ODE trajectories.

3. **Failure Modes** VAEs often suffer from blurry outputs due to the Gaussian decoder likelihood. Consistency models, on the other hand, preserve diffusion's high image fidelity. However, this quality can degrade when the self-consistency constraint is not well-enforced.

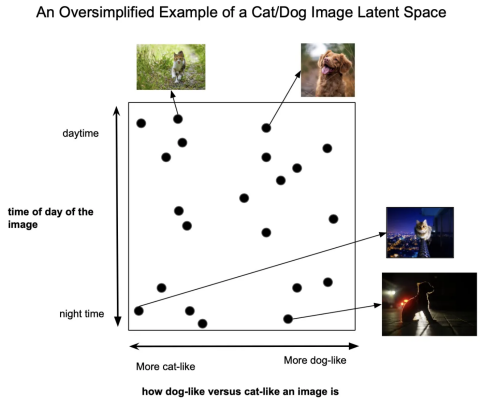

Fig. 3. VAE latent space where each point represents an encoded image. Semantic factors like "cat vs. dog" and "day vs. night" create smooth latent directions that the decoder can invert back into meaningful images. Courtesy of Snell (2021)

.

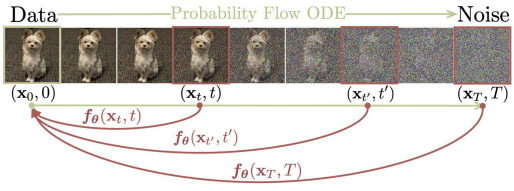

Fig. 4. Illustration of the fundamental mapping learned by a consistency model. Unlike a VAE, which organizes data into a structured latent space, a consistency model learns to map any noisy point along a Probability Flow ODE trajectory back to the same underlying clean image. This many-to-one collapse contrasts with the smooth, structured latent geometry of VAEs. Courtesy of Song et al. (2023)

.

C. Consistency vs. Normalizing Flows. Normalizing flow models and consistency models are both non-adversarial, feed-forward generative models. However, they are very different with respect to invertibility, likelihood, and architecture.

1. **Invertibility**

   Normalizing flows are built around a strict bijective requirement. A flow defines an invertible map

   $$x = f_\theta(z), \qquad z \sim \mathcal{N}(0, I),$$

   where each layer of $f_\theta : \mathbb{R}^d \to \mathbb{R}^d$ must be invertible, and its Jacobian determinant must be tractable. This means that every output corresponds to exactly one latent point and no two different upstream points can map to the same image. Hence, architectures must preserve invertibility at each layer, via affine coupling layers, invertible 1×1 convolutions, and masked autoregressive transforms.

   Consistency models operate under a completely different paradigm. They do not learn an invertible map, but instead a time-parametrized collapse

   $$f_\theta(x_t, t) \mapsto x_\epsilon,$$

   where x_t ranges over an entire PF-ODE trajectory. For any clean data point, x_ϵ, the PF-ODE produces a continuum of noisy states

   $$\mathcal{M}(x_\epsilon) = \{\, x_t : x_t = g_t(x_\epsilon),\ t \in [\epsilon, T] \,\}.$$

   Hence, the consistency model is explicitly trained such that all of these states are mapped back to the same x_ϵ. Thus, the learned mapping is many-to-one in a temporal sense, not invertible and not intended to be.

   The "reverse direction" in consistency models, or going from noise to data, is not obtained by inverting a neural network. During training, the diffusion model follows a deterministic PF-ODE backward in time, and consistency distillation trains f_θ to approximate the endpoint of this flow. At inference, however, the consistency model does not integrate the PF-ODE; it performs a single learned step that directly maps a noisy input to its predicted clean point. Consequently, consistency models do not implement an invertible neural network architecture, but rather a learned projection along the PF-ODE. See Figures 5 and 6. See Figures 5, 6.

2. **Likelihood**

   The flows in normalizing flow models are invertible and differentiable. Hence, we can get exact log-likelihoods using change of variables,

   $$\log p_X(x) = \log p_Z\left(f_\theta^{-1}(x)\right) \ + \ \log \det \left(\frac{\partial x}{\partial f_\theta^{-1}(x)} \right). \tag{15}$$

   Flows are trained using maximum likelihood, which requires computing the Jacobian Determinant and restricting the network architecture to ensure tractability.

   Consistency models, however, do not provide likelihoods. The mapping from noisy image to clean image is many-to-one and thus is not invertible, as discussed. It is trained entirely using consistency losses, and does not use change of variables log-density. This loss does not enforce a likelihood, it enforces pairwise equal outputs for adjacent points on the same PF-ODE trajectory. Hence, consistency models are firmly non-likelihood based generative models.

3. **Architectural Constraints**

   Because of global invertibility requirements, normalizing flow models have extensive architectural constraints. Log determinants and inverses must be efficient to compute. Thus, as dimensionality increases, design freedom and expressivity are restricted. Consistency models, however, have essentially only one architectural constraint, its boundary condition (detailed in Section B). This means that U-Net architectures from DDPMs (Ho et al. (2020)) and EDMs (Karras et al. (2022)), and arbitrary non-linear blocks are permitted. Furthermore, there is no restriction on Jacobian structure and no need for invertible layers, permutations, or special coupling layers. This means that consistency models are much more flexible and expressive per parameter than flows.

4. **Sampling Behavior** For normalizing flows, sampling is trivial, via one forward pass,

$$z \sim \mathcal{N}(0, I), x = f_\theta(z)$$

   Because flows must preserve bijectivity, they cannot project or "denoise" explicitly. They must simply transform the latent Gaussian.

   While consistency models are also only a single forward pass, the semantics are very different.

$$x_0 = f_\theta(x_T, T), x_T \sim \mathcal{N}(0, T^2 I)$$

   In this case, the input x_T is interpreted as maximally noisy data along the PF-ODE trajectory. The model then applies a denoising projection back to a clean manifold. Fundamentally, it is a projection onto the learned data manifold, not a bijection between latent space and image space.

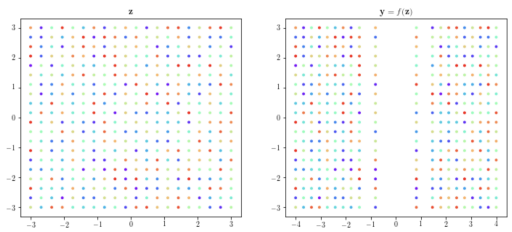

Fig. 5. This illustrates the volume preservation of normalizing flow models. This process is an invertible, smooth diffeomorphism. There is no volume collapse as in consistency models. Courtesy of Abdul-Fatir (2018)

.

4. Implementation and Result Analysis

In addition to theoretical study, I decided to implement a small experiment to compare the sampling behavior of diffusion vs. consistency models in a controlled and unambiguous setting. I used a two-dimensional mixture of Gaussians arranged in a ring as the target data distribution. This is a multimodal, low-dimensional dataset that allows us to directly plot learned trajectories and sample distributions, which is impossible in high-dimensional image spaces. We also precompute a grid of 32 equally spaced times between ϵ and T for use in consistency distillation.

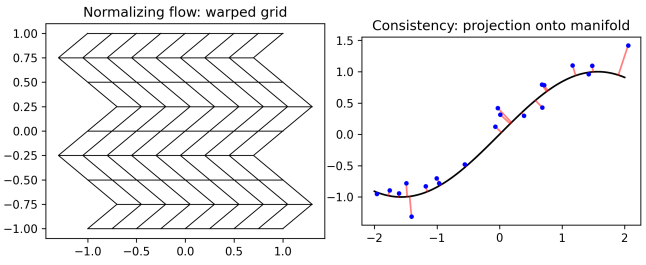

Fig. 6. Geometry of normalizing flows vs. consistency models. Left: an invertible normalizing flow smoothly warps a regular grid in latent space into a deformed grid in data space without folding or collapsing volumes. Right: a consistency-style mapping takes a noisy cloud of samples and explicitly projects them onto a lower-dimensional data manifold, collapsing volumes and breaking invertibility.

The diffusion model is a score-based model trained to learn an approximation to the time-dependent score $\nabla_x \log p_t(x)$. Then, we train a small MLP $s_\phi(x, t)$, which takes as its input a 2D point x and a 64-dimensional sinusoidal time embedding. The time embedding module applies log-spaced sinusoidal features (32 sin and 32 cos components) to the scalar t, followed by a linear projection. This embedding is concatenated with x and passed through two hidden layers, each with width 64, with SiLU activation functions. The final linear layer outputs a 2D score vector.

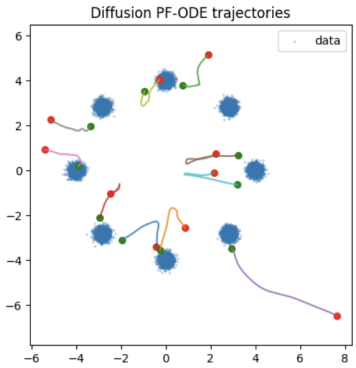

Fig. 7. Deterministic Probability Flow ODE trajectories generated by the trained diffusion model. Each curve shows a backward ODE integration starting from a noisy initialization (red) and flowing toward the underlying clean data distribution (blue), terminating near a denoised endpoint (green). These trajectories illustrate how the Probability Flow ODE gradually removes noise and guides samples toward the data manifold.

We train this network using denoising score matching. For each batch, we sample $t \sim \text{Uniform}([\epsilon, T])$, construct $x_t = x + tz$, and minimize the mean squared error between the network output and analytic score target of $-\frac{x_t - x}{t^2}$. During training, we use the Adam optimizer and a learning rate of 10^{-3}, batch size 512, and 300 epochs. To generate samples, we integrate the Probability Flow ODE,

$$\frac{dx}{dt} = -t\, s_\phi(x, t) \qquad [16]$$

backward in time using 100 Euler steps, starting from $x_T \sim \mathcal{N}(0, T^2 I)$. The resulting trajectories show smooth paths that curve from noisy initial conditions toward the ring of Gaussian

modes, see Figure 7. The final samples match the original, ground truth data distribution relatively well, see Figure 8.

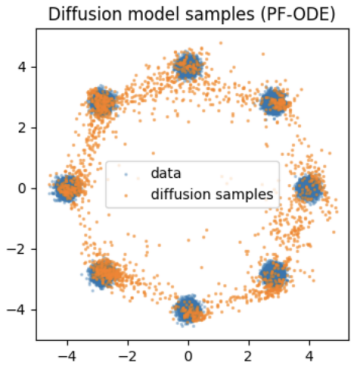

Fig. 8. Final PF-ODE samples (orange) produced by the diffusion model closely match the true data distribution (blue). The model captures both the global ring structure and the local cluster shapes.

The consistency model has a similar general architecture, but is parametrized in skip-connection form appropriate for consistency training. The network $f_\theta(x,t)$ uses the same 64-dimensional time embedding and core MLP $F_\theta(x,t)$ with two SiLU-activated hidden layers of width 64. Hence, the final mapping is defined as

$$f_\theta(x,t) = c_{skip}(t)x + c_{out}(t)F_\theta(x,t) \qquad [17]$$

where

$$c_{skip}(t) = 1 - s; \qquad c_{out}(t) = s; \qquad s = \text{clamp}\left(\frac{t-\epsilon}{T-\epsilon}, 0, 1\right) \qquad [18]$$

As a reminder, this means that f_θ behaves approximately like the identity at low noise ($t = \epsilon$) and gradually shifts toward the learned correction F_θ at higher noise levels. In addition, we maintain an exponential moving average (EMA) copy $f_{\theta'}$ with decay parameter $\mu = 0.999$, which is used as the target network during training and for sampling at test time.

We train the consistency model using Consistency Distillation with the previously trained diffusion model as a fixed teacher. For each batch and for a randomly chosen time index n on the 32-point grid, we sample data x and noise z to create a noisier point $x_{t_{n+1}} = x + t_{n+1}z$. Then, we approximate the preceding PF-ODE state $\hat{x}_{t_n}$ using a single backward Euler step with the teacher's score field. The CD loss enforces self-consistency with the following loss:

$$L_{CD} = \left\| f_\theta(x_{t_{n+1}}, t_{n+1}) - f_{\theta'}(\hat{x}_{t_n}, t_n) \right\|_2^2 \qquad [19]$$

We also added a boundary condition loss that encourages $f_\theta(x_\epsilon, \epsilon) \approx x$ for minimally noisy data points $x_\epsilon = x + \epsilon z'$. Hence, the total objective is

$$L = L_{CD} + L_{bc} \qquad [20]$$

This is optimized using Adam as well. The learning rate is 10^{-3} for 300 epochs. Figure 9 illustrates the resulting behavior, as arrows show how a single forward pass from high-noise samples x_T collapses them directly onto the ring of modes. This contrasts sharply with the many-step PF-ODE trajectories of the diffusion model.

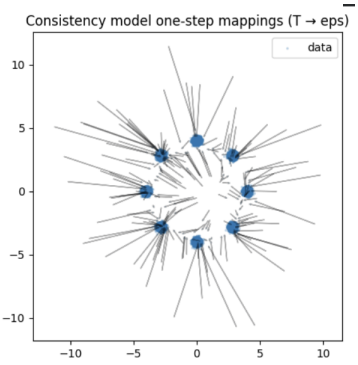

Fig. 9. One-step mappings generated by the consistency model. This demonstrates how high noise samples x_T collapse directly onto the clean data manifold (blue) after a single forward pass. This contrasts with the multistep PF-ODE denoising trajectories required by the diffusion model, seen in Figure 7.

Sampling from the consistency model is extremely simple. We draw $x_T \sim \mathcal{N}(0, T^2 I)$ and fix $t = T$. Then, we compute $x_\epsilon = f_\theta(x_T, T)$ with a single forward pass, see Figure 10 for a visualization of the generations produced by the consistency model. The consistency model also reproduces the ring geometry of the original data distribution, and even clusters around some of the mode locations of the data.

Finally, we measured the sampling cost for 2,000 samples on each model and compared. The diffusion model required 99 network evaluations and took approximately 1.14 seconds. However, the consistency model required a single evaluation and only about 0.018 seconds. This means there was roughly a 100x reduction in function evaluations needed, and 65x reduction in real sampling time. For similar quality generations, this stark difference in sampling and generation time speaks to the utility and strength of consistency models.

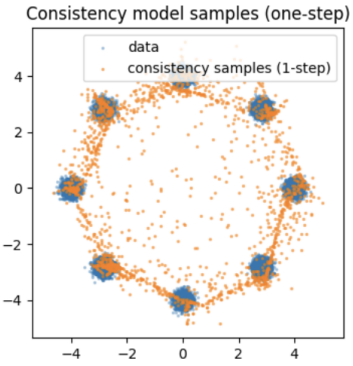

Fig. 10. Consistency model one-step samples (orange) closely match the ring-shaped data distribution (blue). This demonstrates that the model can reproduce the underlying structure using only a single forward pass.

5. Consistency Model Future Work

Even though consistency models offer a paradigm for efficiently generating quality content, there is still room for mechanical improvement, as outlined below (not an exhaustive list):

1. **CD's Dependence on Diffusion Model Teacher**

   Firstly, Consistency Distillation (CD) needs a pretrained diffusion model as its teacher. This means that training a

consistency model requires fully training a base diffusion model, and then there is additional distillation cost to train the consistency model. While consistency models save on sampling and generation compute, they do not save on training compute. Hence, a current goal in the space is to continue to pursue teacher-free Consistency Training (CT), in hopes of making it as strong and stable as Consistency Distillation (CD).

2. **Evaluation Difficulty**

   Similar to diffusion models with implicit samplers, consistency models do not provide tractable log-likelihoods and thus cannot be evaluated with exact likelihood-based metrics as flows can. There is room for development in the space of evaluation metrics for one-step implicit generative models.

3. **Expressivity vs. Training Signal**

   An asset of consistency models is that they allow for very flexible architectures because there are no invertibility constraints. However, besides fulfilling the boundary condition, the only training signal is pairwise consistency between neighboring timesteps. There lacks an explicit likelihood signal to push the model to completely match p_{data}. This quirk raises some questions. Firstly, will the consistency constraint be enough for certain data distributions? Secondly, what does it mean for expressivity when we have the risk of learning "shortcut" mappings that may be sufficient for nearby timestamps, but do not globally match the PF-ODE? Because the loss only enforces that

   $$f_\theta(x_{t_{n+1}}, t_{n+1}) \approx f_{\theta'}(\hat{x}_{t_n}, t_n),$$

   the model is not required to learn how states evolve across the entire interval $[\epsilon, T]$. Instead, it only needs to ensure that adjacent timesteps collapse to the same prediction. As a result, a highly expressive network could learn a degenerate solution in which each small neighborhood in state-time space collapses correctly, but these local solutions do not align coherently in a global approximation of the PF-ODE. Shortcut mappings may ignore the true geometry of the flow, mapping intermediate noise levels directly to visually plausible, but incorrect points on the data manifold. Hence, f_θ may appear correct when examined at closely spaced timesteps, while error significantly when considered across larger temporal gaps.

6. Conclusion

One of the primary limitations of diffusion models is the computational demand associated with up to thousands of neural network inferences during a single generation. However, a consistency model provides a solution to this problem by enforcing a self-consistency constraint along Probability Flow ODE trajectories. The consistency model collapses every noisy variant of a sample back into a single near-clean slate, allowing for high quality generations in a single forward pass (multistep refinement optional). Furthermore, the lack of likelihood-based training objectives and explicit latent variables, among other features, also differentiates consistency models from VAEs and normalizing flow models. However, consistency models still embody the single-step efficiency of these generation models.

Hence, consistency models pose a huge breakthrough in the state-of-the-art, as they produce content that embodies both the fidelity and robustness of diffusion models, but also the computational efficiency of single-step generators.

References

Abdul-Fatir, M. (2018). Normalizing flows, part 1. https://abdulfatir.com/blog/2018/Normalizing-Flows-Part-1/. Accessed: 2025-01-01.

Frans, K., Hafner, D., Levine, S., and Abbeel, P. (2025). One step diffusion via shortcut models.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models.

Karras, T., Aittala, M., Aila, T., and Laine, S. (2022). Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*.

Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes.

Rezende, D. J. and Mohamed, S. (2016). Variational inference with normalizing flows.

Snell, C. (2021). Understanding vq-vae (dall-e explained pt. 1). https://mlberkeley.substack.com/p/vq-vae. Machine Learning at Berkeley Blog, accessed: 2025-01-01.

Song, Y., Dhariwal, P., Chen, M., and Sutskever, I. (2023). Consistency models.