

Diffusion Meets Flow Matching: Two Sides of the Same Coin

Flow matching and diffusion models are two popular frameworks in generative modeling. Despite seeming similar, there is some confusion in the community about their exact connection. In this post, we aim to clear up this confusion and show that *diffusion models and Gaussian flow matching are the same*, although different model specifications can lead to different network outputs and sampling schedules. This is great news, it means you can use the two frameworks interchangeably.

AUTHORS

[Ruiqi Gao](#)

[Emiel Hooeboom](#)

[Jonathan Heek](#)

[Valentin De Bortoli](#)

[Kevin P. Murphy](#)

[Tim Salimans](#)

PUBLISHED

Dec. 2, 2024

AFFILIATIONS

Google DeepMind

Contents

[Overview](#)

[Sampling](#)

[Training](#)

[Diving deeper into samplers](#)

[SDE and ODE perspective](#)

[Closing takeaways](#)



Flow matching has gained popularity recently, due to the simplicity of its formulation and the “straightness” of its induced sampling trajectories. This raises the commonly asked question:

“Which is better, diffusion or flow matching?”

As we will see, diffusion models and flow matching are *equivalent* (for the common special case that the source distribution used with flow matching corresponds to a Gaussian), so there is no single answer to this question. In particular, we will show how to convert one formalism to another. But why does this equivalence matter? Well, it allows you to mix and match techniques developed from the two frameworks. For example, after training a flow matching model, you can use either a stochastic or deterministic sampling method (contrary to the common belief that flow matching is always deterministic).

We will focus on the most commonly used flow matching formalism with the optimal transport path [1], which is closely related to rectified flow [2] and stochastic interpolants [3, 4]. Our purpose is not to recommend one approach over another (both frameworks are valuable, each rooted in distinct theoretical perspectives, and it’s actually even more encouraging that they lead to the same algorithm in practice), but rather to help practitioners understand and feel confident about using these frameworks interchangeably, while understanding the true degrees of freedom one has when tuning the algorithm—regardless of what it’s called.

Check this [Google Colab](#) for code used to produce plots and animations in this post.

Overview

We start with a quick overview of the two frameworks.

Diffusion models

A diffusion process gradually destroys an observed datapoint \mathbf{x} (such as an image) over time t , by mixing the data with Gaussian noise. The noisy data at time t is given by a forward process:

$$\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}). \quad (1)$$

α_t and σ_t define the **noise schedule**. A noise schedule is called variance-preserving if $\alpha_t^2 + \sigma_t^2 = 1$. The noise schedule is designed in a way such that \mathbf{z}_0 is close to the clean data, and \mathbf{z}_1 is close to a Gaussian noise.

To generate new samples, we can “reverse” the forward process: We initialize the sample \mathbf{z}_1 from a standard Gaussian. Given the sample \mathbf{z}_t at time step t , we predict what the clean sample might look like with a neural network (a.k.a. denoiser model) $\hat{\mathbf{x}} = \hat{\mathbf{x}}(\mathbf{z}_t; t)$, and then we project it back to a lower noise level s with the same forward transformation:

$$\mathbf{z}_s = \alpha_s \hat{\mathbf{x}} + \sigma_s \hat{\boldsymbol{\epsilon}}, \quad (2)$$

where $\hat{\boldsymbol{\epsilon}} = (\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}) / \sigma_t$. (Alternatively we can train a neural network to predict the noise $\hat{\boldsymbol{\epsilon}}$.) We keep alternating between predicting the clean data, and projecting it back to a lower noise level until we get the clean sample. This is the DDIM sampler [5]. The randomness of samples only comes from the initial Gaussian sample, and the entire reverse process is deterministic. We will discuss the stochastic samplers later.

Flow matching

In flow matching, we view the forward process as a linear interpolation between the data \mathbf{x} and a noise term $\boldsymbol{\epsilon}$:

$$\mathbf{z}_t = (1 - t)\mathbf{x} + t\boldsymbol{\epsilon}. \quad (3)$$

This corresponds to the diffusion forward process if the noise is Gaussian (a.k.a. Gaussian flow matching) and we use the schedule $\alpha_t = 1 - t, \sigma_t = t$.

Using simple algebra, we can derive that $\mathbf{z}_t = \mathbf{z}_s + \mathbf{u} \cdot (t - s)$ for $s < t$, where $\mathbf{u} = \boldsymbol{\epsilon} - \mathbf{x}$ is the “velocity”, “flow”, or “vector field”. Hence, to sample \mathbf{z}_s given \mathbf{z}_t , we reverse time and replace the vector field with our best guess at time t : $\hat{\mathbf{u}} = \hat{\mathbf{u}}(\mathbf{z}_t; t) = \hat{\boldsymbol{\epsilon}} - \hat{\mathbf{x}}$, represented by a neural network, to get

$$\mathbf{z}_s = \mathbf{z}_t + \hat{\mathbf{u}} \cdot (s - t). \quad (4)$$

Initializing the sample \mathbf{z}_1 from a standard Gaussian, we keep getting \mathbf{z}_s at a lower noise level than \mathbf{z}_t , until we obtain the clean sample.

Comparison

So far, we can already discern the similar essences in the two frameworks:

1. **Same forward process**, if we assume that one end of flow matching is Gaussian, and the noise schedule of the diffusion model is in a particular form.
2. **"Similar" sampling processes**: both follow an iterative update that involves a guess of the clean data at the current time step. (Spoiler: below we will show they are exactly the same!)

Sampling

It is commonly thought that the two frameworks differ in how they generate samples: Flow matching sampling is deterministic with “straight” paths, while diffusion model sampling is stochastic and follows “curved paths”. Below, we clarify this misconception. We will focus on deterministic sampling first, since it is simpler, and will discuss the stochastic case later on.

Imagine you want to use your trained denoiser model to transform random noise into a datapoint. Recall that the DDIM update is given by $\mathbf{z}_s = \alpha_s \hat{\mathbf{x}} + \sigma_s \hat{\boldsymbol{\epsilon}}$. Interestingly, by rearranging terms it can be expressed in the following formulation, with respect to several sets of network outputs and reparametrizations:

$$\tilde{\mathbf{z}}_s = \tilde{\mathbf{z}}_t + \text{Network output} \cdot (\eta_s - \eta_t) \quad (5)$$

Network Output	Reparametrization
$\hat{\mathbf{x}}$ -prediction	$\tilde{\mathbf{z}}_t = \mathbf{z}_t / \sigma_t$ and $\eta_t = \alpha_t / \sigma_t$
$\hat{\boldsymbol{\epsilon}}$ -prediction	$\tilde{\mathbf{z}}_t = \mathbf{z}_t / \alpha_t$ and $\eta_t = \sigma_t / \alpha_t$
$\hat{\mathbf{u}}$ -flow matching vector field	$\tilde{\mathbf{z}}_t = \mathbf{z}_t / (\alpha_t + \sigma_t)$ and $\eta_t = \sigma_t / (\alpha_t + \sigma_t)$

Remember the flow matching update in Equation (4)? This should look similar. If we set the network output as $\hat{\mathbf{u}}$ in the last line and let $\alpha_t = 1 - t, \sigma_t = t$, we have $\tilde{\mathbf{z}}_t = \mathbf{z}_t$ and $\eta_t = t$, which is the flow matching update! More formally, the flow matching update is a Euler sampler of the sampling ODE (i.e., $d\mathbf{z}_t = \hat{\mathbf{u}} dt$), and with the flow matching noise schedule,

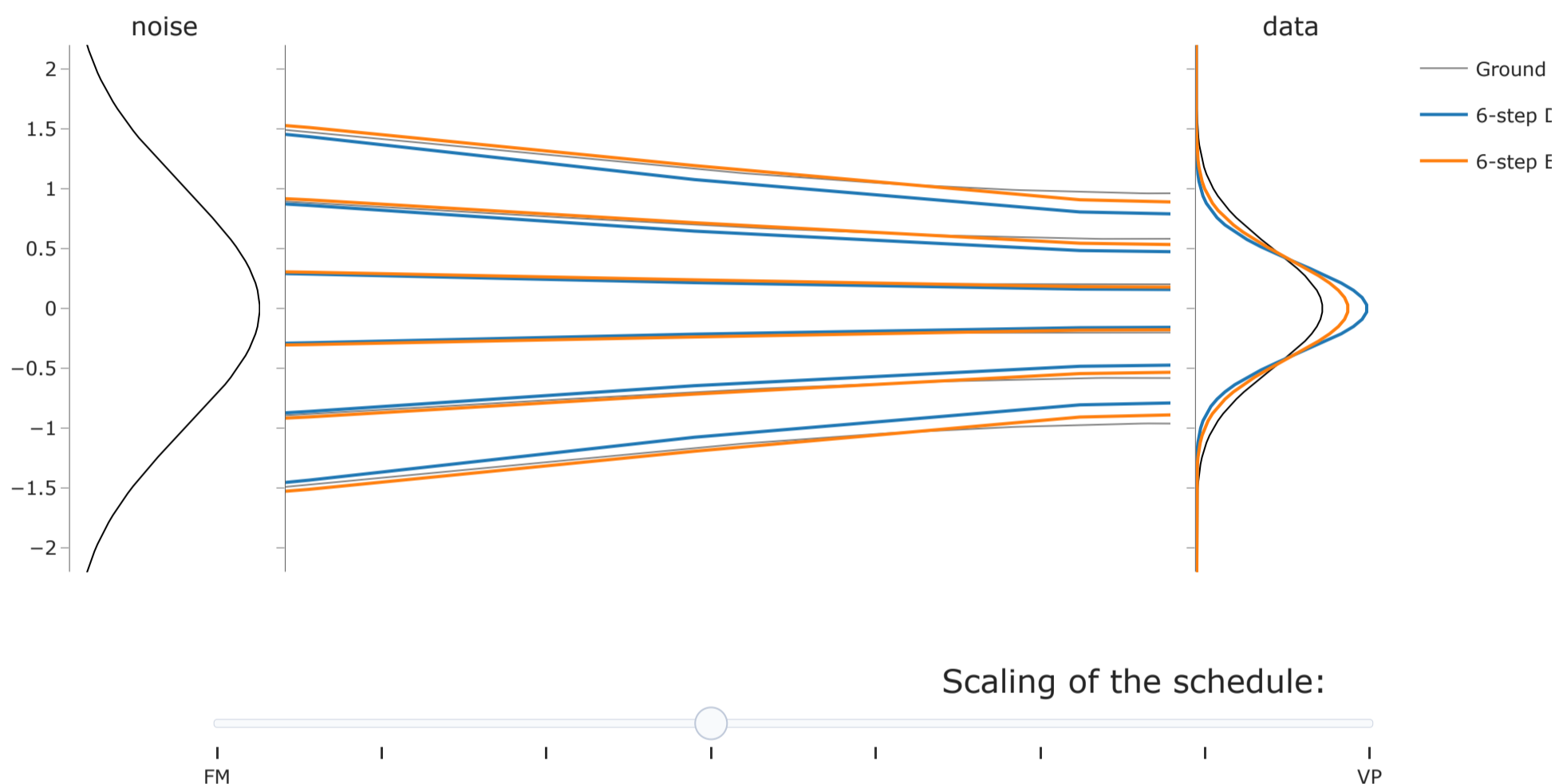
Diffusion with DDIM sampler == Flow matching sampler (Euler).

Some other comments on the DDIM sampler:

1. The DDIM sampler *analytically* integrates the reparametrized sampling ODE (i.e., $d\tilde{\mathbf{z}}_t = [\text{Network output}] \cdot d\eta_t$) if the network output is a *constant* over time. Of course the network prediction is not constant, but it means the inaccuracy of DDIM sampler only comes from approximating the intractable integral of the network output (unlike the Euler sampler of the probability flow ODE [6] which involves an additional linear term of \mathbf{z}_t). The DDIM sampler can be considered a first-order Euler sampler of the reparametrized sampling ODE, which has the same update rule for different network outputs. However, if one uses a higher-order ODE solver, the network output can make a difference, which means the $\hat{\mathbf{u}}$ output proposed by flow matching can make a difference from diffusion models.
2. The DDIM sampler is *invariant* to a linear scaling applied to the noise schedule α_t and σ_t , as scaling does not affect $\tilde{\mathbf{z}}_t$ and η_t . This is not true for other samplers e.g. Euler sampler of the probability flow ODE.

To validate Claim 2, we present the results obtained using several noise schedules, each of which follows a flow-matching schedule ($\alpha_t = 1 - t, \sigma_t = t$) with different scaling factors. Feel free to change the slider below the figure. At the left end, the scaling factor is $\mathbf{1}$, which is exactly the flow matching schedule (FM), while at the right end, the scaling factor is $1/[(1-t)^2 + t^2]$, which corresponds to a variance-preserving schedule (VP). We see that DDIM (and flow matching sampler) always gives the same final data samples, regardless of the scaling of the schedule. The paths bend in different ways as we are showing \mathbf{z}_t (but not $\tilde{\mathbf{z}}_t$), which is scale-dependent along the path. For the Euler sampler of the probability flow ODE, the scaling makes a true difference: we see that both the paths and the final samples change.

Sampling paths with different scalings of the schedule

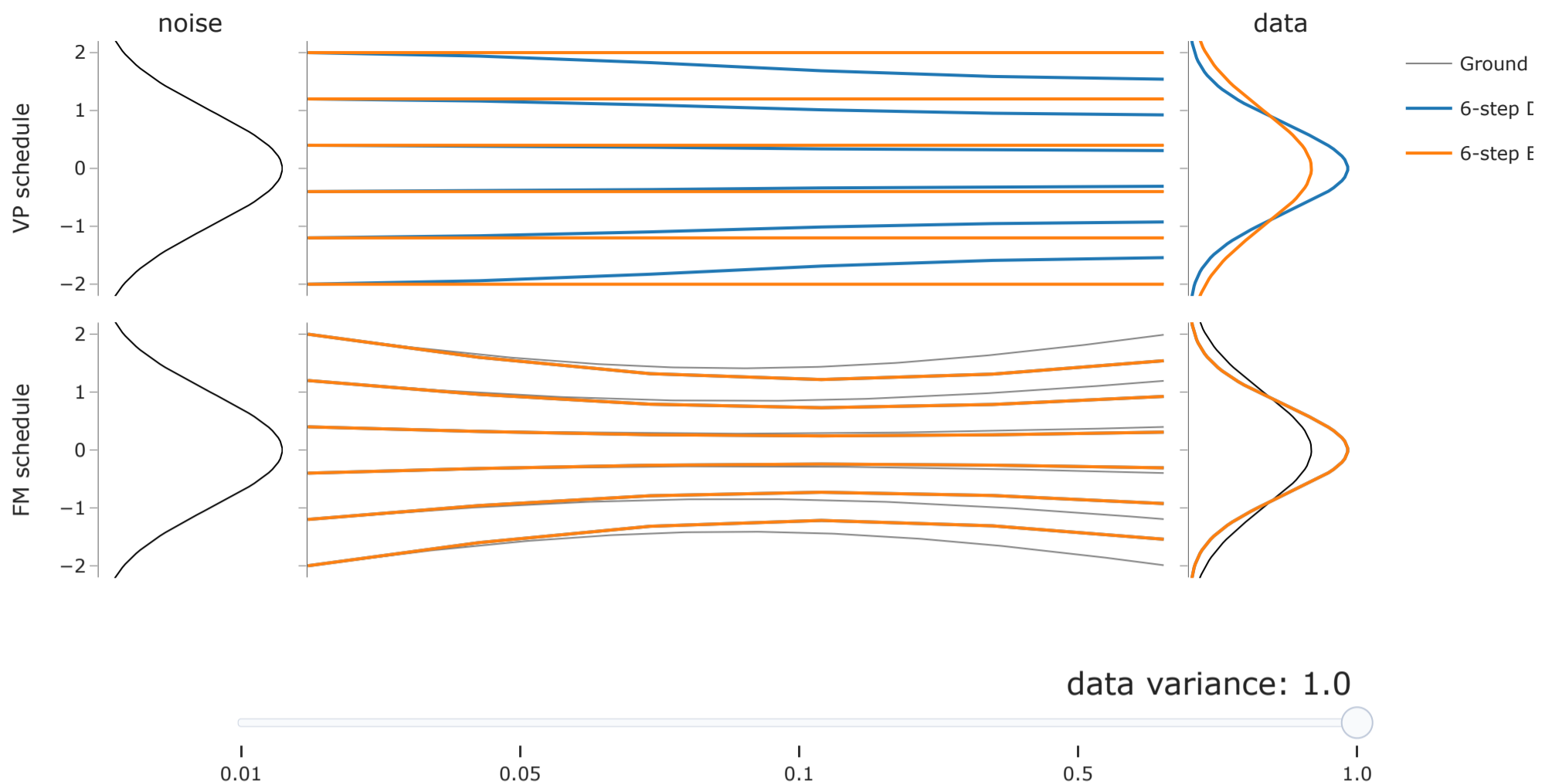


Wait a second! People often say flow matching results in *straight* paths, but in the above figure, the sampling trajectories look *curved*.

Well first, why do they say that? If the model would be perfectly confident about the data point it is moving to, the path from noise to data will be a straight line, with the flow matching noise schedule. Straight line ODEs would be great because it means that there is no integration error whatsoever. Unfortunately, the predictions are not for a single point. Instead they average over a larger distribution. And flowing *straight to a point* != *straight to a distribution*.

In the interactive graph below, you can change the variance of the data distribution on the right hand side by the slider. Note how the variance preserving schedule is better (straighter paths) for wide distributions, while the flow matching schedule works better for narrow distributions.

Variance Preserving vs Flow Matching schedules for varying data distributions



Finding such straight paths for real-life datasets like images is of course much less straightforward. But the conclusion remains the same: The optimal integration method depends on the data distribution.

Two important takeaways from deterministic sampling:

1. **Equivalence in samplers:** DDIM is equivalent to the flow matching sampler, and is invariant to a linear scaling to the noise schedule.
2. **Straightness misnomer:** Flow matching schedule is only straight for a model predicting a single point. For realistic distributions, other schedules can give straighter paths.

Training

Diffusion models [7] are trained by estimating $\hat{\mathbf{x}} = \hat{\mathbf{x}}(\mathbf{z}_t; t)$, or alternatively $\hat{\boldsymbol{\epsilon}} = \hat{\boldsymbol{\epsilon}}(\mathbf{z}_t; t)$ with a neural net. Learning the model is done by minimizing a weighted mean squared error (MSE) loss:

$$\mathcal{L}(\mathbf{x}) = \mathbb{E}_{t \sim \mathcal{U}(0,1), \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})} \left[w(\lambda_t) \cdot \frac{d\lambda}{dt} \cdot \|\hat{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}\|_2^2 \right], \quad (6)$$

where $\lambda_t = \log(\alpha_t^2 / \sigma_t^2)$ is the log signal-to-noise ratio, and $w(\lambda_t)$ is the **weighting function**, balancing the importance of the loss at different noise levels. The term $d\lambda/dt$ in the training objective seems unnatural and in the literature is often merged with the weighting function. However, their separation helps *disentangle* the factors of training noise schedule and weighting function clearly, and helps emphasize the more important design choice: the weighting function.

Flow matching also fits in the above training objective. Recall below is the conditional flow matching objective used by [1, 2] :

$$\mathcal{L}_{\text{CFM}}(\mathbf{x}) = \mathbb{E}_{t \sim \mathcal{U}(0,1), \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})} [\|\hat{\mathbf{u}} - \mathbf{u}\|_2^2] \quad (7)$$

Since $\hat{\mathbf{u}}$ can be expressed as a linear combination of $\hat{\boldsymbol{\epsilon}}$ and \mathbf{z}_t , the CFM training objective can be rewritten as mean squared error on $\boldsymbol{\epsilon}$ with a specific weighting.

How do we choose what the network should output?

Below we summarize several network outputs proposed in the literature, including a few versions used by diffusion models and the one used by flow matching. They can be derived from each other given the current data \mathbf{z}_t . One may see the training objective defined with respect to MSE of different network outputs in the literature. From the perspective of training objective, they all correspond to having some additional weighting in front of the $\boldsymbol{\epsilon}$ -MSE that can be absorbed in the weighting function.

Network Output	Formulation	MSE on Network Output
$\hat{\boldsymbol{\epsilon}}$ -prediction	$\hat{\boldsymbol{\epsilon}}$	$\ \hat{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}\ _2^2$
$\hat{\mathbf{x}}$ -prediction	$\hat{\mathbf{x}} = (\mathbf{x}_t - \sigma_t \hat{\boldsymbol{\epsilon}}) / \alpha_t$	$\ \hat{\mathbf{x}} - \mathbf{x}\ _2^2 = e^{-\lambda} \ \hat{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}\ _2^2$
$\hat{\mathbf{v}}$ -prediction	$\hat{\mathbf{v}} = \alpha_t \hat{\boldsymbol{\epsilon}} - \sigma_t \hat{\mathbf{x}}$	$\ \hat{\mathbf{v}} - \mathbf{v}\ _2^2 = \alpha_t^2 (e^{-\lambda} + 1)^2 \ \hat{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}\ _2^2$
$\hat{\mathbf{u}}$ -flow matching vector field	$\hat{\mathbf{u}} = \hat{\boldsymbol{\epsilon}} - \hat{\mathbf{x}}$	$\ \hat{\mathbf{u}} - \mathbf{u}\ _2^2 = (e^{-\lambda/2} + 1)^2 \ \hat{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}\ _2^2$

In practice, however, the model output might make a difference. For example,

- $\hat{\epsilon}$ -prediction can be problematic at high noise levels, because any error in $\hat{\epsilon}$ will get amplified in $\hat{\mathbf{x}} = (\mathbf{x}_t - \sigma_t \hat{\epsilon}) / \alpha_t$, as α_t is close to 0. It means that small changes create a large loss under some weightings.
- Following the similar reason, $\hat{\mathbf{x}}$ -prediction is problematic at low noise levels, because \mathbf{x} as a target is not informative when added noise is small, and the error gets amplified in $\hat{\epsilon}$.

Therefore, a heuristic is to choose a network output that is a combination of $\hat{\mathbf{x}}$ - and $\hat{\epsilon}$ -predictions, which applies to the $\hat{\mathbf{v}}$ -prediction and the flow matching vector field $\hat{\mathbf{u}}$.

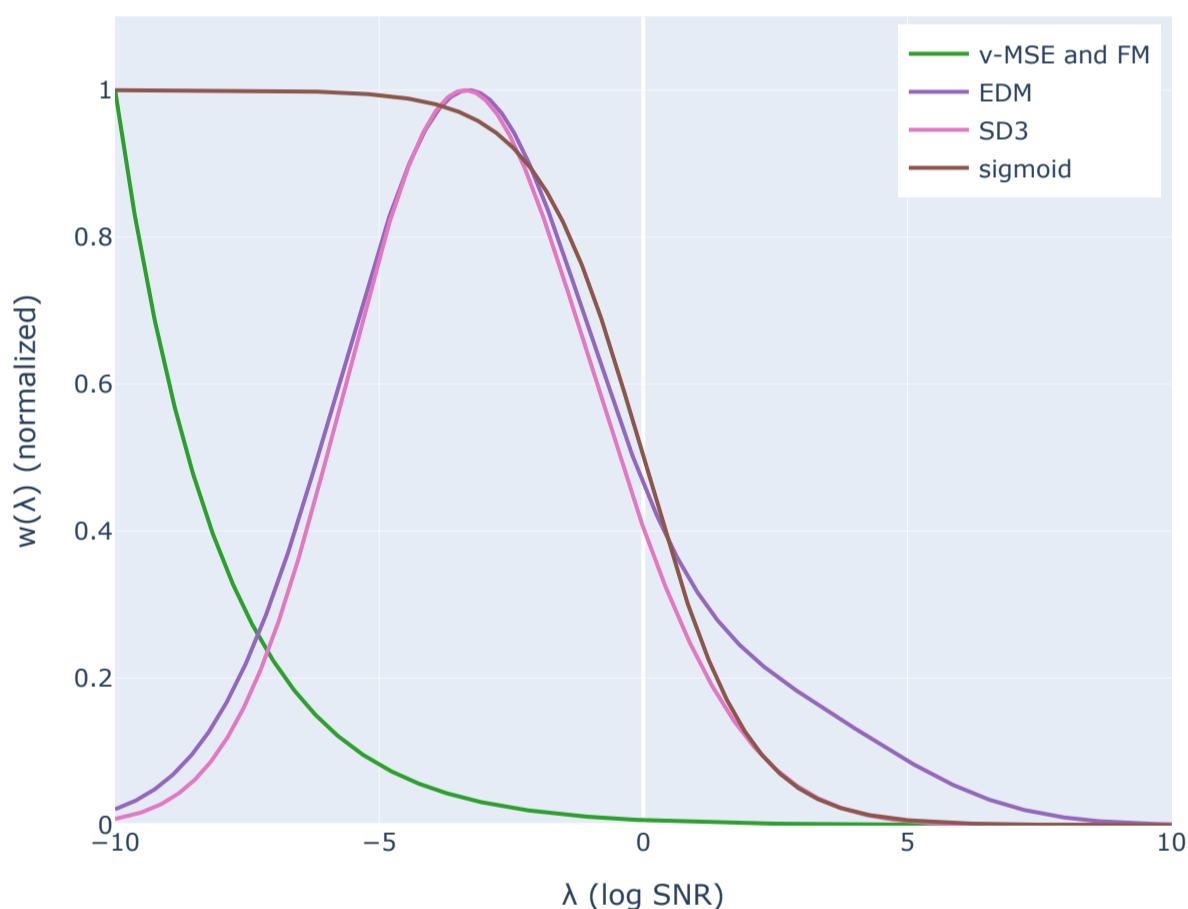
How do we choose the weighting function?

The weighting function is the most important part of the loss. It balances the importance of high frequency and low frequency components in perceptual data such as images, videos and audio [8, 7]. This is crucial, as certain high frequency components in those signals are not perceptible to humans, and thus it is better not to waste model capacity on them when the model capacity is limited. Viewing losses via their weightings, one can derive the following non-obvious result:

Flow matching weighting == diffusion weighting of \mathbf{v} -MSE loss + cosine noise schedule.

That is, the conditional flow matching objective in Equation (7) is the same as a commonly used setting in diffusion models! See Appendix D.2-3 in [7] for a detailed derivation. Below we plot several commonly used weighting functions in the literature, as a function of λ .

Weighting functions



The flow matching weighting (also \mathbf{v} -MSE + cosine schedule weighting) decreases exponentially as λ increases. Empirically we find another interesting connection: The Stable Diffusion 3 weighting [9], a reweighted version of flow matching, is very similar to the EDM weighting [10] that is popular for diffusion models.

How do we choose the training noise schedule?

We discuss the training noise schedule last, as it should be the least important to training for the following reasons:

1. The training loss is *invariant* to the training noise schedule. Specifically, the loss function can be rewritten as $\mathcal{L}(\mathbf{x}) = \int_{\lambda_{\min}}^{\lambda_{\max}} w(\lambda) \mathbb{E}_{\epsilon \sim \mathcal{N}(0, \mathbf{I})} [\|\hat{\epsilon} - \epsilon\|_2^2] d\lambda$, which is only related to the endpoints (λ_{\max} , λ_{\min}), but not the schedule λ_t in between. In practice, one should choose λ_{\max} , λ_{\min} such that the two ends are close enough to the clean data and Gaussian noise respectively. λ_t might still affect the variance of the Monte Carlo estimator of the training loss. A few heuristics have been proposed in the literature to automatically adjust the noise schedules over the course of training. [This blog post](#) has a nice summary.
2. Similar to sampling noise schedule, the training noise schedule is invariant to a linear scaling, as one can easily apply a linear scaling to \mathbf{z}_t and an unscaling at the network input to get the equivalence. The key defining property of a noise schedule is the log signal-to-noise ratio λ_t .
3. One can choose completely different noise schedules for training and sampling, based on distinct heuristics: For training, it is desirable to have a noise schedule that minimizes the variance of the Monte Carlo estimator, whereas for sampling the noise schedule is more related to the discretization error of the ODE / SDE sampling trajectories and the model curvature.

Summary

A few takeaways for training of diffusion models / flow matching:

1. **Equivalence in weightings:** The weighting function is important for training, which balances the importance of different frequency components of perceptual data. Flow matching weightings coincidentally match commonly used diffusion training weightings in the literature.
2. **Insignificance of training noise schedule:** The noise schedule is far less important to the training objective, but can affect the training efficiency.
3. **Difference in network outputs:** The network output proposed by flow matching is new, which nicely balances $\hat{\mathbf{x}}$ - and $\hat{\epsilon}$ -prediction, similar to $\hat{\mathbf{v}}$ -prediction.

Diving deeper into samplers

In this section, we discuss different kinds of samplers in more detail.

Reflow operator

The Reflow operation in flow matching connects noise and data points in a straight line. One can obtain these (data, noise) pairs by running a deterministic sampler from noise. A model can then be trained to directly predict the data given the noise avoiding the need for sampling. In the diffusion literature, the same approach was the one of the first distillation techniques [11].

Deterministic sampler vs. stochastic sampler

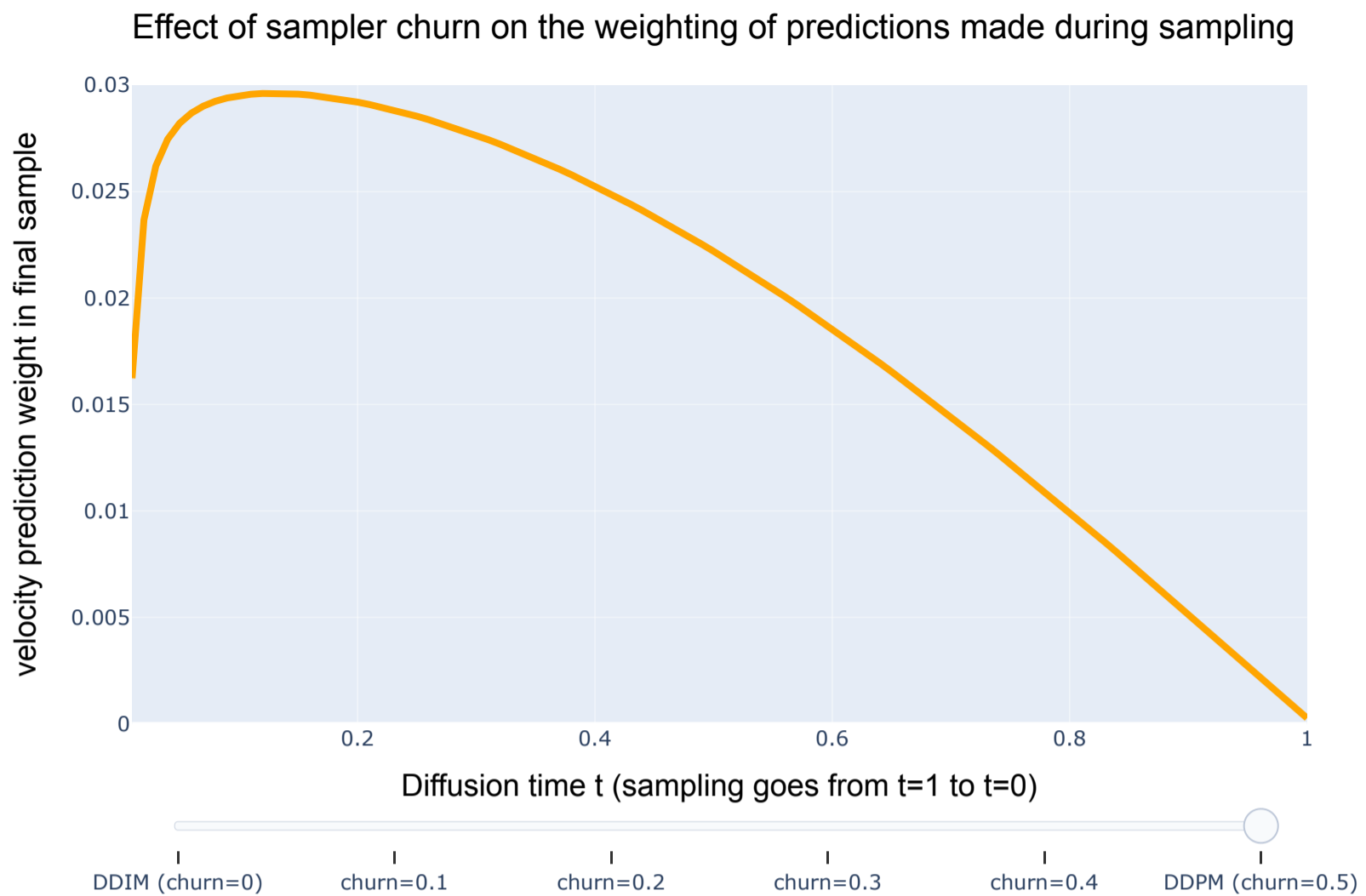
So far we have just discussed the deterministic sampler of diffusion models or flow matching. An alternative is to use stochastic samplers such as the DDPM sampler [12].

Performing one DDPM sampling step going from λ_t to $\lambda_t + \Delta\lambda$ is exactly equivalent to performing one DDIM sampling step to $\lambda_t + 2\Delta\lambda$, and then renoising to $\lambda_t + \Delta\lambda$ by doing forward diffusion. That is, the renoising by doing forward diffusion reverses exactly half the progress made by DDIM. To see this, let's take a look at a 2D example. Starting from the same mixture of Gaussians distribution, we can take either a small DDIM sampling step with the sign of the update reversed (left), or a small forward diffusion step (right):



For individual samples, these updates behave quite differently: the reversed DDIM update consistently pushes each sample away from the modes of the distribution, while the diffusion update is entirely random. However, when aggregating all samples, the resulting distributions after the updates are identical. Consequently, if we perform a DDIM sampling step (without reversing the sign) followed by a forward diffusion step, the overall distribution remains unchanged from the one prior to these updates.

The fraction of the DDIM step to undo by renoising is a hyperparameter which we are free to choose (i.e. does not have to be exact half of the DDIM step), and which has been called the level of *churn* by [10]. Interestingly, the effect of adding churn to our sampler is to diminish the effect on our final sample of our model predictions made early during sampling, and to increase the weight on later predictions. This is shown in the figure below:



Here we ran different samplers for 100 sampling steps using a cosine noise schedule and $\hat{\mathbf{v}}$ -prediction [13]. Ignoring nonlinear interactions, the final sample produced by the sampler can be written as a weighted sum of predictions $\hat{\mathbf{v}}_t$ made during sampling and a Gaussian noise \mathbf{e} : $\mathbf{z}_0 = \sum_t h_t \hat{\mathbf{v}}_t + \sum_t c_t \mathbf{e}$. The weights h_t of these predictions are shown on the y-axis for different diffusion times t shown on the x-axis. DDIM results in an equal weighting of $\hat{\mathbf{v}}$ -predictions for this setting, as shown in [13], whereas DDPM puts more emphasis on predictions made towards the end of sampling. Also see [14] for analytic expressions of these weights in the $\hat{\mathbf{x}}$ - and $\hat{\mathbf{e}}$ -predictions.

SDE and ODE Perspective

We've observed the practical equivalence between diffusion models and flow matching algorithms. Here, we formally describe the equivalence of the forward and sampling processes using ODE and SDE, as a completeness in theory.

Diffusion models

The forward process of diffusion models which gradually destroys a data over time can be described by the following stochastic differential equation (SDE):

$$d\mathbf{z}_t = f_t \mathbf{z}_t dt + g_t d\mathbf{z}, \quad (8)$$

where $d\mathbf{z}$ is an *infinitesimal Gaussian* (formally, a Brownian motion). f_t and g_t decide the noise schedule. The generative process is given by the reverse of the forward process, whose formula is given by

$$d\mathbf{z}_t = \left(f_t \mathbf{z}_t - \frac{1 + \eta_t^2}{2} g_t^2 \nabla \log p_t(\mathbf{z}_t) \right) dt + \eta_t g_t d\mathbf{z}, \quad (9)$$

where $\nabla \log p_t$ is the *score* of the forward process.

Note that we have introduced an additional parameter η_t which controls the amount of stochasticity at inference time. This is related to the *churn* parameter introduced before. When discretizing the backward process we recover DDIM in the case $\eta_t = 0$ and DDPM in the case $\eta_t = 1$.

Flow matching

The interpolation between \mathbf{x} and $\mathbf{\epsilon}$ in flow matching can be described by the following ordinary differential equation (ODE):

$$d\mathbf{z}_t = \mathbf{u}_t dt. \quad (10)$$

Assuming the interpolation is $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \mathbf{\epsilon}$, then $\mathbf{u}_t = \dot{\alpha}_t \mathbf{x} + \dot{\sigma}_t \mathbf{\epsilon}$.

The generative process is simply reversing the ODE in time, and replacing \mathbf{u}_t by its conditional expectation with respect to \mathbf{z}_t . This is a specific case of *stochastic interpolants* [3, 4], in which case it can be generalized to an SDE:

$$d\mathbf{z}_t = (\mathbf{u}_t - \frac{1}{2}\varepsilon_t^2 \nabla \log p_t(\mathbf{z}_t))dt + \varepsilon_t d\mathbf{z}, \quad (11)$$

where ε_t controls the amount of stochasticity at inference time.

Equivalence of the two frameworks

Both frameworks are defined by three hyperparameters respectively: $\mathbf{f}_t, \mathbf{g}_t, \eta_t$ for diffusion, and $\alpha_t, \sigma_t, \varepsilon_t$ for flow matching. We can show the equivalence by deriving one set of hyperparameters from the other. From diffusion to flow matching:

$$\alpha_t = \exp\left(\int_0^t \mathbf{f}_s ds\right), \quad \sigma_t = \left(\int_0^t g_s^2 \exp\left(-2 \int_0^s \mathbf{f}_u du\right) ds\right)^{1/2}, \quad \varepsilon_t = \eta_t g_t.$$

From flow matching to diffusion:

$$\mathbf{f}_t = \partial_t \log(\alpha_t), \quad g_t^2 = 2\alpha_t \sigma_t \partial_t(\sigma_t/\alpha_t), \quad \eta_t = \varepsilon_t / (2\alpha_t \sigma_t \partial_t(\sigma_t/\alpha_t))^{1/2}.$$

In summary, aside from training considerations and sampler selection, diffusion and Gaussian flow matching exhibit no fundamental differences.

Closing takeaways

If you've read this far, hopefully we've convinced you that diffusion models and Gaussian flow matching are equivalent. However, we highlight two new model specifications that Gaussian flow matching brings to the field:

- **Network output:** Flow matching proposes a vector field parametrization of the network output that is different from the ones used in diffusion literature. The network output can make a difference when higher-order samplers are used. It may also affect the training dynamics.
- **Sampling noise schedule:** Flow matching leverages a simple sampling noise schedule $\alpha_t = 1 - t$ and $\sigma_t = t$, with the same update rule as DDIM.

It would be interesting to investigate the importance of these two model specifications empirically in different real world applications, which we leave to future work. It is also an exciting research area to apply flow matching to more general cases where the source distribution is non-Gaussian, e.g. for more structured data like protein [15].

Acknowledgements

Thanks to our colleagues at Google DeepMind for fruitful discussions. In particular, thanks to Sander Dieleman, Ben Poole and Aleksander Hołyński.

References

1. **Flow matching for generative modeling**
Lipman, Y., Chen, R.T., Ben-Hamu, H., Nickel, M. and Le, M., 2022. arXiv preprint arXiv:2210.02747.
2. **Flow straight and fast: Learning to generate and transfer data with rectified flow**
Liu, X., Gong, C. and Liu, Q., 2022. arXiv preprint arXiv:2209.03003.
3. **Building normalizing flows with stochastic interpolants**
Albergo, M.S. and Vanden-Eijnden, E., 2022. arXiv preprint arXiv:2209.15571.
4. **Stochastic interpolants: A unifying framework for flows and diffusions**
Albergo, M.S., Boffi, N.M. and Vanden-Eijnden, E., 2023. arXiv preprint arXiv:2303.08797.
5. **Denoising diffusion implicit models**
Song, J., Meng, C. and Ermon, S., 2020. arXiv preprint arXiv:2010.02502.
6. **Score-based generative modeling through stochastic differential equations**
Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S. and Poole, B., 2020. arXiv preprint arXiv:2011.13456.
7. **Understanding diffusion objectives as the elbo with simple data augmentation**
Kingma, D. and Gao, R., 2024. Advances in Neural Information Processing Systems, Vol 36.
8. **Diffusion is spectral autoregression** [\[HTML\]](#)
Dieleman, S., 2024.
9. **Scaling rectified flow transformers for high-resolution image synthesis**
Esser, P., Kulal, S., Blattmann, A., Entezari, R., Muller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F. and others., 2024. Forty-first International Conference on Machine Learning.
10. **Elucidating the design space of diffusion-based generative models**
Karras, T., Aittala, M., Aila, T. and Laine, S., 2022. Advances in neural information processing systems, Vol 35, pp. 26565–26577.

11. **Knowledge distillation in iterative generative models for improved sampling speed** [\[PDF\]](#)

Luhman, E. and Luhman, T., 2021. arXiv preprint arXiv:2101.02388.

12. **Denoising diffusion probabilistic models**

Ho, J., Jain, A. and Abbeel, P., 2020. Advances in neural information processing systems, Vol 33, pp. 6840–6851.

13. **Progressive Distillation for Fast Sampling of Diffusion Models**

Salimans, T. and Ho, J., 2022. International Conference on Learning Representations.

14. **Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models**

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C. and Zhu, J., 2022. arXiv preprint arXiv:2211.01095.

15. **Se (3)-stochastic flow matching for protein backbone generation**

Bose, A.J., Akhound-Sadegh, T., Huguet, G., Fatras, K., Rector-Brooks, J., Liu, C., Nica, A.C., Korablyov, M., Bronstein, M. and Tong, A., 2023. arXiv preprint arXiv:2310.02391.


For attribution in academic contexts, please cite this work as

```
@inproceedings{gao2025diffusionmeetsflow,  
  author = {Gao, Ruiqi and Hoogetboom, Emiel and Heek, Jonathan and Bortoli, Valentin De and Murphy, Kevin P. and Salimans, Tim},  
  title = {Diffusion Meets Flow Matching: Two Sides of the Same Coin},  
  year = {2024},  
  url = {https://diffusionflow.github.io/}  
}
```

0 Comments - powered by [utteranc.es](#)

Write Preview

Sign in to comment

 Styling with Markdown is supported

Sign in with GitHub