Faster Training of Diffusion Models and Improved Density Estimation via Parallel Score Matching

Etrit Haxholli, Marco Lorenzi

Faster Training of Diffusion Models and Improved Density Estimation via Parallel Score Matching

Etrit Haxholli*
Epione Research Group
Inria, Université Cote d'Azur
etrit.haxholli@inria.fr

Marco Lorenzi
Epione Research Group
Inria, Université Cote d'Azur
marco.lorenzi@inria.fr

Abstract

In Diffusion Probabilistic Models (DPMs), the task of modeling the score evolution via a single time-dependent neural network necessitates extended training periods and may potentially impede modeling flexibility and capacity. To counteract these challenges, we propose leveraging the independence of learning tasks at different time points inherent to DPMs. More specifically, we partition the learning task by utilizing independent networks, each dedicated to learning the evolution of scores within a specific time sub-interval. Further, inspired by residual flows, we extend this strategy to its logical conclusion by employing separate networks to independently model the score at each individual time point. As empirically demonstrated on synthetic and image datasets, our approach not only significantly accelerates the training process by introducing an additional layer of parallelization atop data parallelization, but it also enhances density estimation performance when compared to the conventional training methodology for DPMs.

1 Introduction

Forward Diffusion Processes (FDPs) represent a category of Markov chains that gradually transform the data distribution into a standard multivariate normal one by incrementally corrupting the data samples. Within the context of Diffusion Probabilistic Models (DPMs), one aims to train a neural network in order to mimic the reverse dynamics of this process (Sohl-Dickstein et al., 2015).

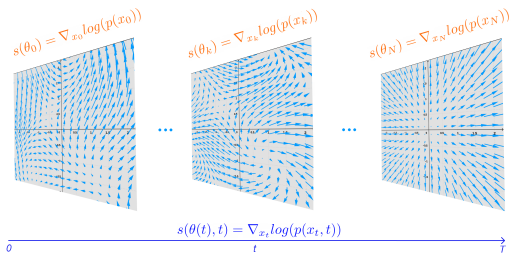

Figure 1: Instead of using a time dependent network to learn the scores for all times (below: blue), at each time point we can use a model per score (above: orange), naturally allowing parallel training of diffusion models.

Allowing the number of degradation steps to tend to infinity enables the FDP to be depicted as a Stochastic Differential Equation (SDE) (Song et al., 2021). This SDE defines a distinct distribution at every temporal point of the diffusion process. Given samples from the data distribution, it is possible to efficiently generate samples from the distribution at any specified time t_i, which can then be utilized to train a neural network to approximate the score at time t_i through the denoising (Vincent, 2011; Ho et al., 2020) or the sliced score matching loss (Hyvärinen, 2005; Song et al., 2020). The common approach (Ho et al., 2020; Song et al., 2021; Rombach et al., 2022) is to train a single time-varying neural network to model all scores, i.e., to model the evolution of the score (Figure 1). More detailed

*The code is available at: https://github.com/ehaxholli/PSM

Neural Information Processing Systems (NeurIPS) 2023 Workshop on Diffusion Models.

background material is provided in Appendix E.
To expedite training and increase model flexibility, we leverage the intrinsic property of DPMs that allows scores at different time points to be optimized independently. Consequently, instead of training a single time-varying U-Net to model the scores of the distributions at all time points, we partition the training task by dividing the integration time interval of the SDE into smaller sub-intervals. For each sub-interval, we employ a time-dependent U-Net to model the evolution of scores for the distributions defined within that sub-interval. This strategy is referred to as time-varying parallel score matching (TPSM). Each such U-Net corresponds to a CNF block within the CNF framework, and their composition provides the complete evolution of distribution as determined by the FDP.
With an increasing number of such sub-intervals, the task assigned to each U-Net is simplified, and in the limit, each time sub-interval converges to a point. Consequently, in addition to the previously described strategy, we train a multitude of smaller networks, which are not time-dependent, such that each network learns the score at a single time point of the diffusion process (Figure 1). This methodology is designated as Discrete Parallel Score Matching (DPSM).
We evaluate our Parallel Score Matching (PSM) methodologies on 2D data, in addition to image datasets such as CIFAR-10, CelebA 64×64, and ImageNet 64×64, (Liu et al., 2015; Deng et al., 2009). The findings attest that our approach not only enhances the log-likelihood results, but also expedites the training process, without incurring penalties related to memory or inference time.

2 PSM Framework and Relation to Piece-wise Continuous Flows

In the context of continuous normalizing flows, the loss function is computationally costly, as not only it contains the end point of the integration, but also the integral of the divergence along that integration path. Furthermore, additional computational and GPU memory constraints also surface when employing multiple CNF blocks within a piece-wise continuous flow model, as each CNF block introduces a new set of parameters. Given that these transformations are interconnected in a chain, they must coexist in memory during the maximum likelihood optimization procedure. Moreover, in all cases when continuous normalizing flows are trained via likelihood maximization, the time-dependent networks are obliged to generate an evolution process for the data distribution that culminates in a standard normal distribution.
Contrarily, diffusion models parameterized by time-dependent neural networks ameliorate the complexity of learning the evolution process, since the FDP defines the time-based evolution of the data distribution, simplifying the framework's task to merely modeling the already defined vector fields:

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)[\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p_t(\boldsymbol{x}(t))]dt;\ t \in [0,1] \text{ and } b(t) = 10t. \tag{1}$$

However, if parameterized by a single time-varying network, these models still grapple with limited flexibility due to a single set of parameters (model) being tasked with modeling all the vector fields defined by the FDP. Moreover, the approximation of a score at time t is dependent on the approximation of other scores at various time points, which hampers the speed of the training process. This standard approach (SA) in Diffusion Probabilistic Models (DPMs) is outlined in Algorithm 3 in Appendix E for comparison, and is abbreviated as SA-DPM. As shown in Appendix D, the scores in a diffusion process evolve continuously, with the given PDE dynamics:

$$\frac{\partial s(\boldsymbol{x},t)}{\partial t} = \frac{1}{2}b(t)\nabla_{\boldsymbol{x}} tr\frac{\partial s(\boldsymbol{x},t)}{\partial \boldsymbol{x}} + \frac{1}{2}b(t)\frac{\partial s(\boldsymbol{x},t)}{\partial \boldsymbol{x}}(\boldsymbol{x} + s(\boldsymbol{x},t)). \tag{2}$$

Given that the distribution at each time t_i and its associated score are intrinsically determined by the data distribution and the Forward Diffusion Process (FDP), we can expedite training by concurrently learning the scores $\nabla_{\boldsymbol{x}_{t_i}} \log p(\boldsymbol{x}_{t_i}, t_i)$ through each $s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_{t_i})$ at all times t_i. If our models proficiently approximate these scores, the transformation resulting from combining these models will be smooth. This instigates partitioning the diffusion process, specifically, the interval $t \in [0,1]$ is divided into $\bigcup_{i=0}^{N-1}[t_i, t_{i+1}]$. Thereby, it remains feasible to train a time-dependent neural network $s_{\boldsymbol{\theta_i}}(\boldsymbol{x}_t, t)$ to acquire scores $\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t, t)$ for time $t \in [t_i, t_{i+1}]$. Such an approach diminishes the complexity of tasks for each model, bolstering modeling capability, whilst preserving time continuity. Moreover, given the parallelizability of training, a solitary model is loaded in memory per device during training, and post-training it can be stored on a disk and recalled at testing. The training procedure of this framework is described in Algorithm 1, Appendix A. In this case, each score modeling network $s_{\boldsymbol{\theta_i}}(\boldsymbol{x}_t, t)$ corresponds to a CNF block in the continuous normalizing flow representation.

Elevating this approach to its apex, the duration of each interval approaches 0, thus we utilize a single network to model the score per time-point. This framework corresponds to an invertible residual flow with infinitesimal scaling, as invertibility and differentiability are guaranteed by the fact that the learned transformation approximates the diffusion process. A description of this training procedure is given in Algorithm 2, Appendix A.
A comparison of the properties of Parallel Score Matching (PSM) with other generative diffeomorphism-based frameworks can be found in Table 4, Appendix A. Details about the generation procedure and likelihood estimation in the PSM framework can also be found in Appendix A.

3 Experiments

We contrast the density estimation performance of DPMs trained via parallel score matching and those trained through the standard approach (SA-DPM). We conduct experiments on 2D data of toy distributions, alongside standard benchmark datasets including CIFAR-10, CelebA, and ImageNet. At no stage do we employ data parallelization, which constitutes an auxiliary parallelization strategy that complements the methodology presented in this paper. More detailed information about the experiments such as hyper-parameters and network architectures can be found in Appendix H. Limitations and future prospects are discussed in Appendix I.

Table 1: A comparison of the results between SA-DPM and TPSM variants. The results are given in NLL (lower is better). Parallel training time is given in parentheses (blocks $\times$ hours per block).

Data	SA-DPM	TPSM$_A$	TPSM$_B$	TPSM$_C$
TY	0.76 (1$\times$ **4h**)	0.69 (2$\times$ **1h**)	0.64 (4$\times$ **1h**)	0.58 (200$\times$ **1h**)
HG	1.11 (1$\times$ **1.3h**)	1.07 (2$\times$ **0.3h**)	1.04 (4$\times$ **0.3h**)	1.03 (200$\times$ **0.3h**)

3.1 Toy 2D Datasets

To visually discern the distribution modeling capabilities between the baseline and the TPSM approach, we initially test both frameworks on 2D toy data. The two distributions we examine are TY (refer to the upper row of Figure 2) and HG (lower row of Figure 2).

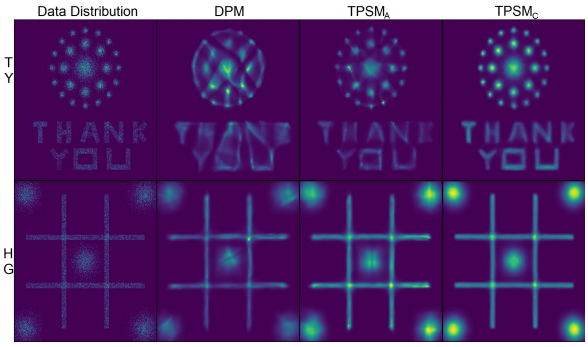

Figure 2: Probability density modeling capabilities of SA (second column), TPSM$_A$ (third column) and TPSM$_C$ (fourth column) on 2D data of toy distributions. The performance of TPSM$_B$ is shown in Appendix G.

The time-varying network deployed is identical in both methods and for both distributions. In SA-DPM the model attempts to learn all the scores of the evolution of the distribution dictated by the FDP (Equation (1)). In contrast, for the TPSM$_A$ approach, we train two distinct models (networks), with the initial model spanning the time interval $[0, 0.1]$ and the subsequent one covering $[0.1, 1]$. In the case of TPSM$_B$, we partition the diffusion process into four subintervals with the following splits $\{0,\ 0.02,\ 0.1,\ 0.3,\ 1\}$. Conversely, in the extreme TPSM$_C$ approach, we utilize 200 such networks, where network i is trained to learn the scores corresponding to the evolution of the distribution dictated by the identical process, constrained to the time interval $t \in [\frac{i}{200}, \frac{i+1}{200}]$. As evidenced in Figure 2, TPSM demonstrates a substantially enhanced performance in the evaluated tasks, when juxtaposed with the baseline (SA-DPM), notwithstanding the fact that the latter underwent a training duration four times lengthier. In Table 1, we present the results of the test NLL for both models across both datasets, in addition to the difference in training time. The RK4 solver with 1000 steps (4000 function evaluations) is used during inference. The

results of additional experiments on 2D data where the Flow-Matching setting, (Lipman et al., 2023), is used instead of DDPM are provided in Appendix F.

3.2 Image Datasets

We parameterize the SA-DPM model using the time-varying U-Net architecture introduced in (Ho et al., 2020) and set the number of channels to 64. The same network is utilized in each block for the TPSM approach. The initial TPSM variant explored, designated as TPSM0, follows a similar approach as TPSM$_A$ in the synthetic 2D data case, wherein the diffusion interval $[0, 1]$ is partitioned into two subintervals: $[0, 0.1]$ and $[0.1, 1]$. By training each network in each block for half the number of parameter updates compared to the SA-DPM network, the training time is reduced by 50% through parallelization. Moreover, improvements in likelihood estimation results are observed (Table 2), with no adverse effects on inference time or memory usage, as illustrated in Tables 7, 8, and 9 in Appendix B. During the inference phase, 100 steps were executed within the interval $[0, 0.1]$ and 900 steps within $[0.1, 1]$. As in the case of 2D data, the RK4 solver was used. Moreover, we evaluate TPSM1,

Table 2: The results of TPSM0, where the two time subintervals have unequal length. In parentheses we give the number of GPUs (blocks) $\times$ the time of training per block. In square brackets we give the number of GPUs (blocks) $\times$ the number of parameter updates per block.

Method	CIFAR-10	CelebA 64×64	ImageNet 64×64
SA-DPM	3.13 (1× **72h**) [1×**800k**]	2.06 (1× **72h**) [1×**800k**]	3.62 (1× **180h**) [1×**2000k**]
TPSM0	3.12 (2× **36h**) [2×**400k**]	1.92(2× **36h**) [2×**400k**]	3.55 (2× **90h**) [2×**1000k**]

which consists of 10 blocks, and to investigate the limiting behavior of TPSM, we introduce TPSM2, comprised of 100 blocks. For TPSM1, block i models the score associated with times $t \in [\frac{i}{10}, \frac{i+1}{10}]$, whereas in the case of TPSM2, this changes to $t \in [\frac{i}{100}, \frac{i+1}{100}]$. The results are given in Table 3 and demonstrate significant improvements in density estimation and training time through parallelization. The network implemented in the DPSM framework differs significantly, as it does not depend on time. In this instance, we elect to employ the initial U-Net as introduced and implemented in the pioneering U-Net paper (Ronneberger et al., 2015). The discretization process incorporates 1000 steps, signifying the use of 1000 basic U-Nets, each per step. The classical non time-varying U-Net exhibits approximately 3.5 times faster training time per parameter update compared to the time-varying U-Net. Coupled with fewer parameter updates, this accelerates DPSM training, making it up to 50 times quicker than that of SA-DPM. Additionally, the U-Net's reduced complexity facilitates usage of batch sizes that are four times larger than in the time-varying scenario. However, the DPSM implementation necessitates utilizing interpolation techniques for density estimation (Appendix H). Details on the number of parameter updates for each model and dataset are available in Table 6, Appendix B. Generated samples from all models can be found in Appendix C. In Appendix B, we provide results when SA-DPM and TPSM2 are trained for the same number of parameter updates.

Table 3: Results comparing the performance of SA-DPM and PSM approaches. We test the models on image data. The results are given in bits/dim (lower is better). In parentheses we give the number of GPUs (blocks) $\times$ the time of training per block.

Method	CIFAR-10	CelebA 64×64	ImageNet 64×64
SA-DPM	3.13 (1× **72h**)	2.06 (1× **72h**)	3.62 (1× **180h**)
TPSM1	3.11 (10× **9h**)	2.07 (10× **9h**)	3.60 (10× **22h**)
TPSM2	2.93 (100× **4.5h**)	1.90 (100× **4.5h**)	3.55 (100× **14h**)
DPSM	2.93 (1000× **1.5h**)	1.94 (1000× **2.5h**)	3.59 (1000× **7h**)

4 Conclusion

We have introduced strategies for Parallel Score Matching in training diffusion probabilistic models. By leveraging the characteristics of diffusion models that allow for individual score modeling, we have significantly reduced training durations and enhanced model performance.

Acknowledgments and Disclosure of Funding

This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002. The authors are grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support.

References

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., & Jacobsen, J.-H. (2019). Invertible residual networks. In K. Chaudhuri, & R. Salakhutdinov (Eds.) *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 of *Proceedings of Machine Learning Research*, (pp. 573–582). PMLR.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, (pp. 248–255). Ieee.

Dinh, L., Krueger, D., & Bengio, Y. (2015). NICE: non-linear independent components estimation. In Y. Bengio, & Y. LeCun (Eds.) *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.) *Advances in Neural Information Processing Systems*, vol. 33, (pp. 6840–6851). Curran Associates, Inc.

Hyvärinen, A. (2005). Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, *6*(24), 695–709.

Kingma, D. P., Salimans, T., Poole, B., & Ho, J. (2021). On density estimation with diffusion models. In A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.) *Advances in Neural Information Processing Systems*.

Kobyzev, I., Prince, S. J., & Brubaker, M. A. (2021). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(11), 3964–3979.

Lipman, Y., Chen, R. T. Q., Ben-Hamu, H., Nickel, M., & Le, M. (2023). Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*.

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, *22*(57), 1–64.

Rezende, D., & Mohamed, S. (2015). Variational inference with normalizing flows. In F. Bach, & D. Blei (Eds.) *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37 of *Proceedings of Machine Learning Research*, (pp. 1530–1538). Lille, France: PMLR.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 10684–10695).

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.) *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, (pp. 234–241). Cham: Springer International Publishing.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In F. Bach, & D. Blei (Eds.) *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37 of *Proceedings of Machine Learning Research*, (pp. 2256–2265). Lille, France: PMLR.

Song, Y., Garg, S., Shi, J., & Ermon, S. (2020). Sliced score matching: A scalable approach to density and score estimation. In R. P. Adams, & V. Gogate (Eds.) *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, vol. 115 of *Proceedings of Machine Learning Research*, (pp. 574–584). PMLR.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*.

Tabak, E. G., & Turner, C. V. (2013). A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, *66*(2), 145–164.

Tabak, E. G., & Vanden-Eijnden, E. (2010). Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, *8*(1), 217 – 233.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Computation*, *23*(7), 1661–1674.

Wang, P. (2020). https://github.com/lucidrains/denoising-diffusion-pytorch.

A Additional Details Regarding the Framework (TPSM and DPSM)

In this section of the Appendix, we provide the algorithms for TPSM and DPSM (Algorithms 1 and 2). Additionally, Table 4 provides a comparative analysis of the characteristics of Normalizing Flows and Diffusion Models. This juxtaposition underscores the advantages enabled by the PSM method, leveraging attributes of diffusion models that are not utilized in the standard approach. In the end of this section we provide additional details about image generation and likelihood estimation in TPSM and DPSM.

Algorithm 1
Time-varying Parallel Score Matching (TPSM)

Input: data $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, batch-size p
Split $[0, 1]$ into $\cup_{i=0}^{N-1}[t_i, t_{i+1}]$
for $i = 0$ **to** $N-1$, in parallel **do**
  Train model $\boldsymbol{s}_{\boldsymbol{\theta}_i}$
  repeat
    uniformly sample τ_1, ..., τ_p from $[t_i, t_{i+1}]$
    sample $\boldsymbol{x}^0_{\pi(1)}$, ..., $\boldsymbol{x}^0_{\pi(p)}$ from $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$
    generate $\boldsymbol{x}^{\tau_1}_{\pi(1)}$, ..., $\boldsymbol{x}^{\tau_p}_{\pi(p)}$ using Eq. 29
    minimize $\Sigma_j ||\boldsymbol{\varepsilon}_{\boldsymbol{\theta}_i}(\boldsymbol{x}^{\tau_j}_{\pi(j)}(\boldsymbol{\varepsilon_j}), \tau_j) - (\boldsymbol{\varepsilon_j})||^2$
  until convergence
end for

Algorithm 2
Discrete Parallel Score Matching (DPSM)

Input: data $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, batch-size p
Discretize $[0, 1]$ into $\{t_0 = 0, t_1, ..., t_N = 1\}$
for $i = 1$ **to** N, in parallel **do**
  Train model $\boldsymbol{s}_{\boldsymbol{\theta}_i}$
  repeat
    uniformly sample t_i from $\{t_0, ..., t_N\}$
    sample $\boldsymbol{x}^0_{\pi(1)}$, ..., $\boldsymbol{x}^0_{\pi(p)}$ from $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$
    generate $\boldsymbol{x}^{t_i}_{\pi(1)}$, ..., $\boldsymbol{x}^{t_i}_{\pi(p)}$ using Eq. 29
    minimize $\Sigma_j ||\boldsymbol{\varepsilon}_{\boldsymbol{\theta}_i}(\boldsymbol{x}^{t_i}_{\pi(j)}(\boldsymbol{\varepsilon_j})) - (\boldsymbol{\varepsilon_j})||^2$
  until convergence
end for

Table 4: A comparison of the properties of Parallel Score Matching (PSM) approaches with other generative diffeomorphism-based frameworks.

METHOD:	CNF	SA-DPM	PSM-DPM
COMPUTATIONALLY DEMANDING LOSS	√	×	×
MEMORY BOTTLENECK	√	×	×
LIMITED VARIABILITY IN TIME	√	√	×
NECESSITY TO DEVISE A DIFFEOMORPHISM	√	×	×
SCORE OPTIMIZATION INTERDEPENDENCY	√	√	×

Image Generation and Likelihood Estimation in TPSM and DPSM

Regarding the time-varying Parallel Score Matching (TPSM) approach, we can generate data as in the original framework by iterating the following integration process:

$$\boldsymbol{x}(t_i) = \boldsymbol{x}(t_{i+1}) + \int_{t_i+1}^{t_i} \big[-\frac{1}{2} b(\tau)\big(\boldsymbol{x}(\tau) + s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_\tau, \tau)\big)\big] d\tau, \tag{3}$$

and perform likelihood estimation via

$$\log p(\boldsymbol{x}(t_0)) = \log p(\boldsymbol{x}(t_N)) + \Sigma_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} \big[-\frac{1}{2} b(\tau) tr \frac{\partial\big(\boldsymbol{x}(\tau) + s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_\tau, \tau)\big)}{\partial \boldsymbol{x}(\tau)} \big] d\tau, \tag{4}$$

where $\boldsymbol{s}_{\boldsymbol{\theta}_i}(\boldsymbol{x}_t, t) = -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}_i}(\boldsymbol{x}_t, t)}{\sqrt{1 - e^{-\int_0^t b(s) ds}}}$.

Similarly, in the case of Discrete Parallel Score Matching (DPSM), we can easily generate data via back-integrating

$$\boldsymbol{x}(t_i) = \boldsymbol{x}(t_{i+1}) - \epsilon\big[-\frac{1}{2} b(t_{i+1})\big(\boldsymbol{x}(t_{i+1}) + s_{\boldsymbol{\theta}_{t_{i+1}}}(\boldsymbol{x}_{t_{i+1}})\big)\big] \tag{5}$$

Likewise, we can perform likelihood estimation as follows

$$\log p(\boldsymbol{x}(t_0)) = \log p(\boldsymbol{x}(t_N)) + \Sigma_{i=1}^{N}\big[-\epsilon\frac{1}{2}b(t_i)tr\frac{\partial\big(\boldsymbol{x}(t_i)+s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_{t_i})\big)}{\partial\boldsymbol{x}(t_i)}\big]. \tag{6}$$

In addition, in both approaches data samples can be generated by integrating the corresponding reverse SDE (Song et al., 2021).

$$d\boldsymbol{x}(t) = -b(t)[\frac{1}{2}\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)}\log p_t(\boldsymbol{x}(t))]dt + \sqrt{b(t)}d\boldsymbol{w}. \tag{7}$$

B Additional Results

Below we give additional results with regards to the experimental section.

First, we set up a comparison between TPSM2 and SA-DPM, where the number of parameter updates for each network is the same in both approaches. To elaborate, each network in TPSM2 undergoes 50k parallel parameter updates, and the network in SA-DPM also experiences 50k parameter updates. Given this configuration, the training duration for both approaches is equivalent. However, as displayed in Table 5, the disparity in their performance is quite significant.

Table 5: Results comparing the performance of SA-DPM and TPSM2 for the same number of parameter updates. We test the models on CIFAR-10, CelebA, and ImageNet. The results are given in bits/dim (lower is better), the number of parameter updates is given in parentheses, and the training time is given in square brackets.

Method:	CIFAR-10	CelebA 64×64	ImageNet 64×64
SA-DPM	3.30 (50k) [4.5H]	2.38 (50k) [4.5H]	3.76 (150k) [14H]
TPSM2	2.93 (50k) [4.5H]	1.90 (50k) [4.5H]	3.55 (150k) [14H]

Table 6: Results related to Table 3, comparing number of parameter updates of SA-DPMs and the parallel score matching approaches, namely TPSM and DPSM.

Method:	CIFAR-10	CelebA	ImageNet
SA-DPM	800k	800k	2000k
TPSM0	400k	400k	1000k
TPSM1	100k	100k	250k
TPSM2	50k	50k	150k
DPSM	50k	100k	250k

Table 7: Inference results on ImageNet. Inference time is given in seconds while memory usage is provided in MB.

Task	Generation SDE	Generation ODE	Likelihood Estimation
SA-DPM	131.8s (2396MB)	674.4s (4260MB)	858.8s (7164MB)
TPSM0	119.8s (2334MB)	635.7s (3951MB)	936.2s (6221MB)
TPSM1	123.6s (2396MB)	626.8s (2800MB)	887.7s (6624MB)
TPSM2	137.1s (2381MB)	567.3s (2392MB)	792.4s (6187MB)
DPSM	124.7s (2709MB)	326.2s (2569MB)	335.2s (2670MB)

Furthermore, we present inferential outcomes in relation to generation and likelihood estimation time, as well as the memory utilization of all the considered approaches. It is noteworthy that PSM approaches not only abstain from inducing inferential penalties, but also have the potential to expedite generation and likelihood estimation in certain cases, such as DPSM. The batch size for SDE and ODE generation was consistently maintained at 32, whereas for likelihood estimation, it

was uniformly reduced to 16. In all instances, the generation of SDEs was executed employing 1000 Numerical Function Evaluations (NFE). In contrast, for ODE generation and likelihood estimation pertaining to SA-DPM and TPSM, a total of 1000 steps employing the RK4 method were taken, equating to 4000 function evaluations. Specifically, in the context of DPSM, a total of 4000 steps were performed using interpolation and integration via the Euler method for both ODE generation and likelihood estimation. The results are provided in Tables 7, 8 and 9.

Table 8: Inference results on CelebA. Inference time is given in seconds while memory usage is provided in MB.

Task	Generation SDE	Generation ODE	Likelihood Estimation
SA-DPM	121.2s (2450MB)	662.8s (4156MB)	892.2s (7139MB)
TPSM0	116.3s (2402MB)	599.3s (3704MB)	933.9s (6261MB)
TPSM1	126.9s (2426MB)	661.4s (2789MB)	929.3s (6470MB)
TPSM2	146s (2435MB)	605.5s (2401MB)	837.8s (616.8MB)
DPSM	116.6s (2518MB)	303.1s (2529MB)	318.8s (2650MB)

Table 9: Inference results on CIFAR-10. Inference time is given in seconds while memory usage is provided in MB.

Task	Generation SDE	Generation ODE	Likelihood Estimation
SA-DPM	38.3s (1890MB)	184.7s (2279MB)	380.2s (3528MB)
TPSM0	38.5s (1780MB)	195.2s (2252MB)	354s (2906MB)
TPSM1	37.9s (1900MB)	198.3s (1989MB)	393.8s (2964MB)
TPSM2	51.1s (1883MB)	185.6s (1895MB)	367s (2920MB)
DPSM	95.2s (1930MB)	171.3s (1959MB)	217.8s (1959MB)

C Additional Generated Samples

Below we provide generated samples from all the models that are present in Table 3.

Figure 3: Random non cherry-picked generated CIFAR-10 images. Top-Left: SA-DPM, 800k parameter updates, 1 CNF block, 10k sampling steps via the reverse SDE. Top-Right: TPSM, 100k parameter updates, 10 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Left: TPSM, 50k parameter updates, 100 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Right: DPSM, 50k parameter updates, 1000 CNF blocks, 1k sampling steps via the reverse SDE.

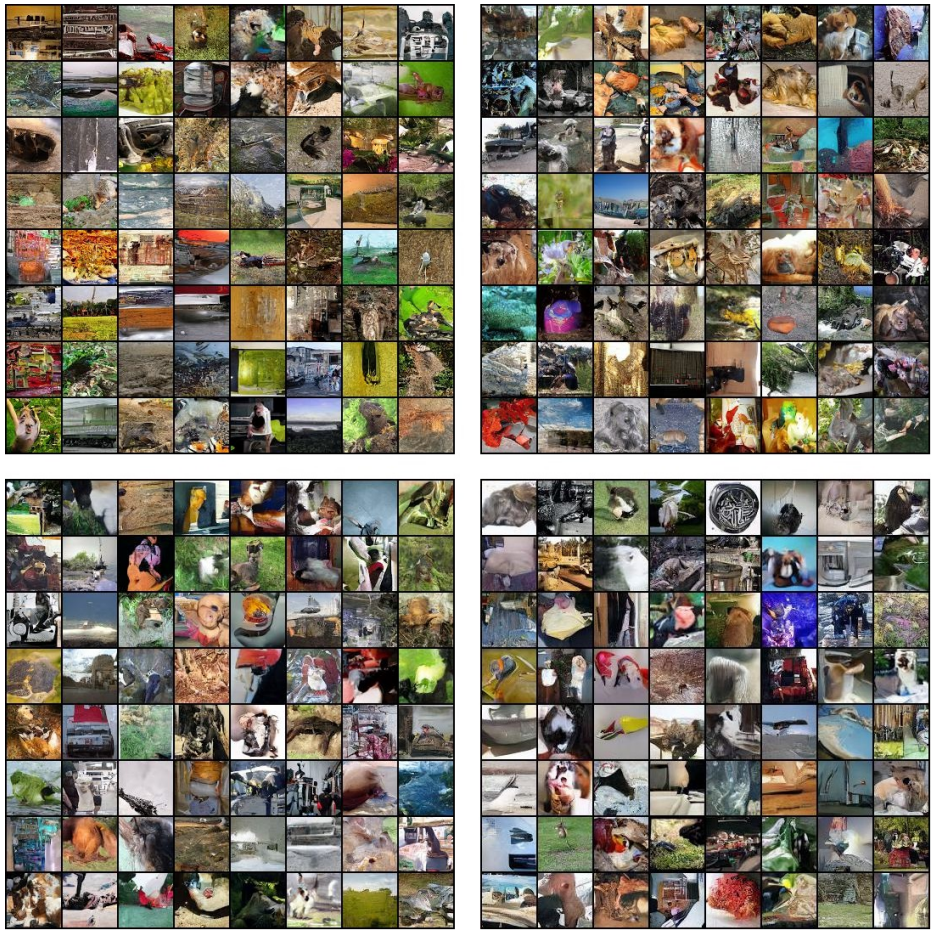

Figure 4: Random non cherry-picked generated ImageNet 64x64 images. Top-Left: SA-DPM, 2000k parameter updates, 1 CNF block, 10k sampling steps via the reverse SDE. Top-Right: TPSM, 250k parameter updates, 10 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Left: TPSM, 150k parameter updates, 100 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Right: DPSM, 250k parameter updates, 1000 CNF blocks, 1k sampling steps via the reverse SDE.

Figure 5: Random non cherry-picked generated CelebA images. Top-Left: SA-DPM, 800k parameter updates, 1 CNF block, 10k sampling steps via the reverse SDE. Top-Right: TPSM, 100k parameter updates, 10 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Left: TPSM, 50k parameter updates, 100 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Right: DPSM, 100k parameter updates, 1000 CNF blocks, 1k sampling steps via the reverse SDE.

Figure 6: Random non cherry-picked generated CelebA, ImageNet64x64 and Cifar-10 images generated by TPSM0 with 2 CNF blocks and 10000 reverse SDE steps. The number of parameter updates is half that of SA-DPM per block as given in Table 6.

D The Continuous Evolution of the Score During Diffusion

The SDE process of transforming the data distribution to a normal one is the following:

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)\boldsymbol{x}(t)dt + \sqrt{b(t)}d\boldsymbol{w}. \tag{8}$$

The equivalent process in terms of the evolution of the distribution is given by the following ODE:

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)\big(\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)\big)dt = f(\boldsymbol{x}(t), t)dt. \tag{9}$$

The instantaneous change of variable formula states that,

$$\frac{d \log p(\boldsymbol{x}(t), t)}{dt} = -tr\frac{\partial f(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}, \tag{10}$$

thus by substituting f:

$$\frac{d \log p(\boldsymbol{x}(t), t)}{dt} = \frac{1}{2}b(t)tr\frac{\partial \boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)} = \frac{1}{2}b(t)D + \frac{1}{2}b(t)tr\frac{\partial \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}. \tag{11}$$

It is easy to notice that,

$$\frac{d\nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{dt} = \nabla_{\boldsymbol{x}(t)}\frac{d \log p(\boldsymbol{x}(t), t)}{dt} = \frac{1}{2}b(t)\nabla_{\boldsymbol{x}(t)}tr\frac{\partial \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}. \tag{12}$$

If we denote $s(\boldsymbol{x}(t), t) = \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)$, the last expression becomes:

$$\frac{ds(\boldsymbol{x}(t), t)}{dt} = \frac{1}{2}b(t)\nabla_{\boldsymbol{x}(t)}tr\frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}. \tag{13}$$

Since,

$$\frac{ds(\boldsymbol{x}(t), t)}{dt} = \frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}\frac{d\boldsymbol{x}(t)}{dt} + \frac{\partial s(\boldsymbol{x}(t), t)}{\partial t} \tag{14}$$

using equation 13, we get:

$$\frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}\frac{d\boldsymbol{x}(t)}{dt} + \frac{\partial s(\boldsymbol{x}(t), t)}{\partial t} = \frac{1}{2}b(t)\nabla_{\boldsymbol{x}(t)}tr\frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}, \tag{15}$$

and therefore

$$\frac{\partial s(\boldsymbol{x}, t)}{\partial t} = \frac{1}{2}b(t)\nabla_{\boldsymbol{x}}tr\frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} - \frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{dt}. \tag{16}$$

We conclude that the score evolves continuously in time as described by the following PDE:

$$\frac{\partial s(\boldsymbol{x}, t)}{\partial t} = \frac{1}{2}b(t)\nabla_{\boldsymbol{x}}tr\frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} + \frac{1}{2}b(t)\frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}}(\boldsymbol{x} + s(\boldsymbol{x}, t)). \tag{17}$$

This emphasizes the fact that the scores of all the intermediate distributions defined by the FDP are completely determined by the score of the initial (data) distribution.

E Background and Related Work

E.1 Normalizing Flows (NFs)

Normalizing Flows: A Normalizing Flow (Tabak & Vanden-Eijnden, 2010; Tabak & Turner, 2013; Rezende & Mohamed, 2015; Dinh et al., 2015) is a transformation defined as a sequence of diffeomorphisms that converts a base probability distribution (e.g., a standard normal) into another distribution by warping the domain on which they are defined. Let $\boldsymbol{Z}$ be a random variable, and define $\boldsymbol{X} = g(\boldsymbol{Z})$, where g is a diffeomorphism with inverse h. If we denote their probability density functions by $f_{\boldsymbol{Z}}$ and $f_{\boldsymbol{X}}$, the change of variable theorem states that:

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = f_{\boldsymbol{Z}}(\boldsymbol{z})|\frac{d\boldsymbol{z}}{d\boldsymbol{x}}| = f_{\boldsymbol{Z}}(h(\boldsymbol{x}))|\frac{dh(\boldsymbol{x})}{d\boldsymbol{x}}|. \tag{18}$$

The objective is the optimization of parameters of h to maximize the likelihood of sampled data points $\boldsymbol{x}$. After training, one can input any test point $\boldsymbol{x}$ on the RHS of Equation 18, and calculate its likelihood. For the generative task, being able to easily recover g from h is essential, as the generated point $\boldsymbol{x}_g$ will take form $\boldsymbol{x}_g$=$g(\boldsymbol{z}_s)$, where $\boldsymbol{z}_s$ is a sampled point from the base distribution $f_{\boldsymbol{Z}}$.
For increased modeling flexibility, we can use a chain (flow) of transformations, $\boldsymbol{z}_i = g_i(\boldsymbol{z}_{i-1}), i \in [n]$. The interested reader can find a more in-depth review of normalizing flows in (Kobyzev et al., 2021) and (Papamakarios et al., 2021).

Neural ODE Flows: Neural ODEs (Chen et al., 2018) are continuous generalizations of residual networks:

$$\begin{aligned} \boldsymbol{x}_{t_{i+1}} &= \boldsymbol{x}_{t_i} + \epsilon f(\boldsymbol{x}_{t_i}, t_i, \boldsymbol{\theta}) \\ \rightarrow \boldsymbol{x}(t) &= \boldsymbol{x}(0) + \int_0^t f(\boldsymbol{x}(\tau), \tau, \boldsymbol{\theta}) d\tau, \text{ as } \epsilon \rightarrow 0, \end{aligned} \tag{19}$$

where for a network with finite weights, injectivity is guaranteed by the Picard–Lindelöf theorem. In (Chen et al., 2018), the expression for the instantaneous change of variable is derived, which enables one to train continuous normalizing flows and perform likelihood estimation:

$$\log p(\boldsymbol{x}(0)) = \log p(\boldsymbol{x}(T)) + \int_0^T tr\frac{\partial f(\boldsymbol{x}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{x}(t)} dt, \tag{20}$$

where $\boldsymbol{x}(0)$ represents a sample from the data. Such models are better known as Continuous Normalizing Flows (CNFs).

Invertible Residual Flows (IRFs): These models (Behrmann et al., 2019) are similar to the discretized version of Continuous Normalizing Flows:

$$\boldsymbol{x}_{t_{i+1}} = \boldsymbol{x}_{t_i} + f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta}_{\boldsymbol{t_i}})$$

It can be noticed that in this case the output of the residual block is not scaled before being added to the input, and furthermore, each block is comprised of a different set of trainable parameters, instead of allowing the behaviour of the model to evolve via a time input. As injectivity cannot be guaranteed via the Picard–Lindelöf theorem, bijectivity has to be enforced by imposing the contraction condition $Lip(f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta}_{\boldsymbol{t_i}})) < 1$, where $Lip(f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta}_{\boldsymbol{t_i}}))$ is the Lipschitz constant of $f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta}_{\boldsymbol{t_i}})$.

Denoising Diffusion Models: A forward diffusion process or diffusion process (Sohl-Dickstein et al., 2015) is a fixed Markov chain that gradually adds Gaussian noise to the data according to a schedule $\{b_t | t \in [n]\}$:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) := \mathcal{N}(\boldsymbol{x}_{t-1}\sqrt{1-b_t}, b_t\boldsymbol{I}). \tag{21}$$

Such a process transforms the data distribution into a standard multivariate normal distribution. If we denote $a_t = 1 - b_t$ and $\bar{a}_t = \prod_{s=1}^{t} a_s$, then we can write:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_0\sqrt{\bar{a}_t}, (1-\bar{a}_t)\boldsymbol{I}). \tag{22}$$

The reverse process conditioned on the initial sample is also described by a chain of Gaussian distributions:

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}(\mu(\boldsymbol{x}_t, \boldsymbol{x}_0), \boldsymbol{\Sigma}(t)), \tag{23}$$

where

$$\mu(\boldsymbol{x}_t, \boldsymbol{x}_0) = \mu(\boldsymbol{x}_t, \boldsymbol{x}_0(\boldsymbol{x}_t, \boldsymbol{\varepsilon})) = \frac{1}{\sqrt{a_t}}\big(\boldsymbol{x}_t - \frac{b_t}{\sqrt{1-\bar{a}_t}}\boldsymbol{\varepsilon}\big),$$
$$\boldsymbol{\Sigma}(t) = b_t \boldsymbol{I}.$$

The goal is to approximate this reverse process via

$$p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) := \mathcal{N}(\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}(t)) \tag{24}$$

that converts the standard multivariate normal distribution into the data distribution. To this end, for each step t, the KL divergence between $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$ and $p(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ is minimized, which amounts to minimizing

$$\mathbb{E}_{\boldsymbol{x_0}, \boldsymbol{x_t}} ||\mu_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) - \mu(\boldsymbol{x}_0, \boldsymbol{x}_t)||^2, \tag{25}$$

or equivalently

$$\mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{\varepsilon}} ||\boldsymbol{\varepsilon}_\theta(\boldsymbol{x_0}\sqrt{\bar{a}_t} + \sqrt{(1-\bar{a}_t)}\boldsymbol{\varepsilon}, t) - \boldsymbol{\varepsilon}||^2, \tag{26}$$

for $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, data samples $\boldsymbol{x_0}$, and a neural network $\boldsymbol{\varepsilon}_\theta(\boldsymbol{x_t}, t)$ (Ho et al., 2020).

E.2 SDE Diffusion Models and their CNF Representation

For an $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, the forward diffusion process defined in Equation 21 can be written as

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1}\sqrt{1-b_t} + \sqrt{b_t}\boldsymbol{\varepsilon}. \tag{27}$$

As derived in (Song et al., 2021), the continuous counterpart of this process takes the form

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)\boldsymbol{x}(t)dt + \sqrt{b(t)}d\boldsymbol{w}. \tag{28}$$

In this case Equation 22 becomes

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_0\mu_t, \sigma_t^2\boldsymbol{I}), \tag{29}$$

where $\mu_t = e^{-\frac{1}{2}\int_0^t b(s)ds}$ and $\sigma_t = \sqrt{(1 - e^{-\int_0^t b(s)ds})}$.

As shown through the Fokker-Plack equation, the evolution of the probability density function of the data, as dictated by the FDP, is identical to the evolution dictated by the following ODE transformation:

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)[\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p_t(\boldsymbol{x}(t))]dt = f_t(\boldsymbol{x}(t))dt. \tag{30}$$

Knowing f_t allows us to perform data generation and likelihood estimation, through the framework of continuous normalizing flows. It can be observed that the only unknown in f_t, is the score $\nabla_{\boldsymbol{x}(t)} \log p_t(\boldsymbol{x}(t))$. This quantity can be modelled using a neural network $\boldsymbol{s}_\theta(\boldsymbol{x}(t), t)$ trained either through sliced score matching (Song et al., 2020):

$$min\ \mathbb{E}_{\boldsymbol{x}_t \sim p(\boldsymbol{x}_t)}[\frac{1}{2}||\boldsymbol{s}_\theta(\boldsymbol{x}_t, t)||^2 + div(\boldsymbol{s}_\theta(\boldsymbol{x}_t, t))] \tag{31}$$

or the equivalent MSE denoising loss (Ho et al., 2020; Kingma et al., 2021):

$$min\ \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{\varepsilon}} ||\boldsymbol{s}_\theta(\boldsymbol{x}_0\mu_t + \sigma_t\boldsymbol{\varepsilon}, t) - (-\frac{\boldsymbol{\varepsilon}}{\sigma_t})||^2. \tag{32}$$

Algorithm 3 Standard Approach in Diffusion Models

Input: data $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, batch-size p
Train model $\boldsymbol{s_\theta}(t)$
repeat
  sample τ_1, ..., τ_p from $[0, 1]$
  sample $\boldsymbol{x}^0_{\pi(1)}$, ..., $\boldsymbol{x}^0_{\pi(p)}$ from $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$
  generate $\boldsymbol{x}^{\tau_1}_{\pi(1)}$, ..., $\boldsymbol{x}^{\tau_p}_{\pi(p)}$ using Equation 29
  minimize $\Sigma_j ||\boldsymbol{\varepsilon_\theta}(\boldsymbol{x}^{\tau_j}_{\pi(j)}(\boldsymbol{\varepsilon_j}), \tau_j) - (\boldsymbol{\varepsilon_j})||^2$.
until convergence

F DDPM and Flow Matching on 2D toy data

Beyond the investigations delineated in the main paper's experimental section, we conducted additional experiments involving the utilization of 10 blocks that evenly distribute the diffusion process time interval. These experiments were executed within the DDPM framework, which is consistently utilized throughout the paper, but also in this special case in the novel Flow-Matching framework.

Table 10: A comparison of the properties of SA-DPM with $\text{TPSM}_{10blocks}$, in the DDPM framework. The results are given in negative log-likelihood (lower is better).

Method:	SA-DPM	$\text{TPSM}_{10blocks}$
TY	0.76 (1× **4H**)	0.69 (10× **1H**)
HG	1.11 (1× **1.3H**)	1.05 (10× **0.3H**)

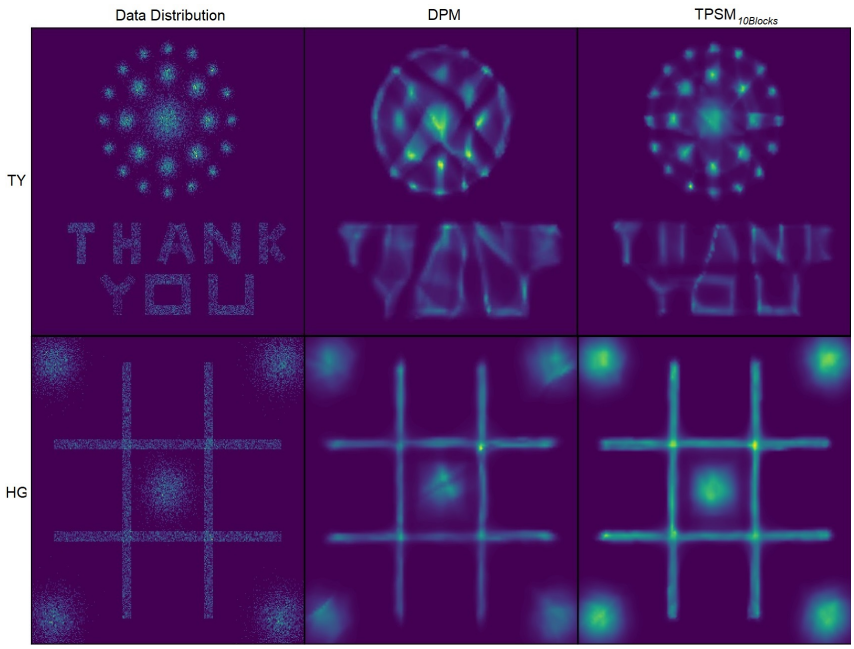

Figure 7: SA-DPM and TPSM with 10 blocks comparison on 2D toy data when the DDPM framework is used.

Within the DDPM framework, the majority of score (probability) evolution occurs when t approaches 0. Consequently, the outcomes of $\text{TPSM}_{10Blocks}$ hold potential for substantial enhancement if more recent settings were employed, such as Flow-Matching, as elucidated in (Lipman et al., 2023), wherein the evolution of the score is more uniformly distributed over time. This improvement is demonstrated in Table 11 and Figure 8. Our observations indicate that the implementation of the Flow-Matching framework bolsters the effectiveness of all approaches. However, the TPSM results exhibit a more pronounced enhancement in the context of the TY data, while in the case of HG the improvements of SA-DPM and TPSM are similar. Thus, even within the confines of more contemporary and efficient frameworks, our parallel score matching strategy continues to deliver the advantageous outcomes postulated in the paper.

Table 11: A comparison of the properties of SA-DPM with TPSM$_{10blocks}$, in the OT Flow-Matching framework. The results are given in negative log-likelihood (lower is better).

Method:	SA-DPM	TPSM$_{10blocks}$
TY	0.75 (1× **4H**)	0.66 (10× **1H**)
HG	1.10 (1× **1.3H**)	1.04 (10× **0.3H**)

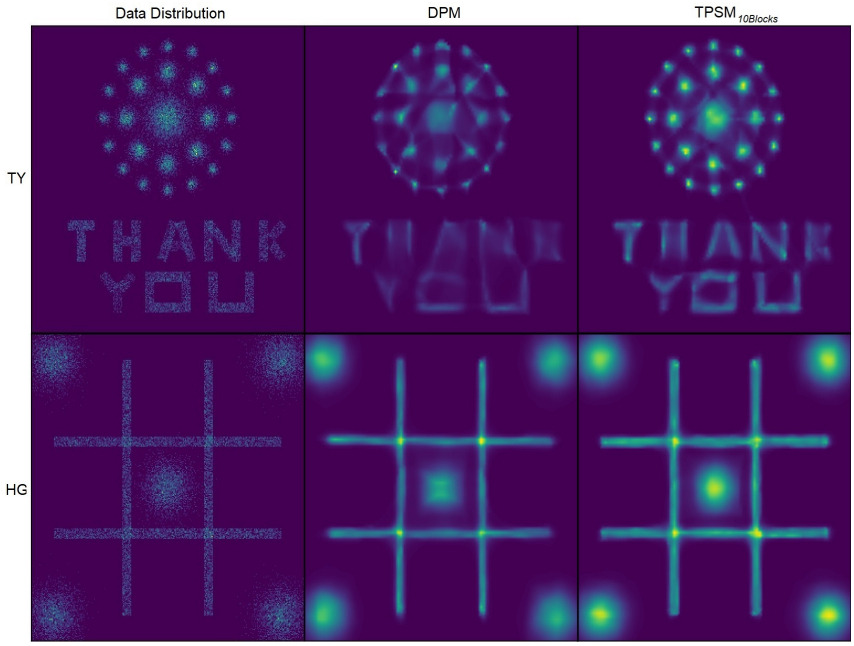

Figure 8: SA-DPM and TPSM with 10 blocks comparison on 2D toy data when the OT Flow-Matching framework is used.

G TPSM-B Visual Results

Owing to the limitations on available space within the primary manuscript, we present here the visual results of TPSM$_B$, which utilizes 4 neural networks. We notice that even if we train these four networks sequentially, the total training time is the same as in SA-DPM, while the resulting performance shows significant improvements.

H Additional Experimental Details

For the 2D toy distribution data scenario, we compared the TPSM method against the baseline, namely, SA-DPM. The batch size across all experiments is 512, and we employ a learning rate of 10^{-3}. Each network was trained on a single CPU of equivalent performance and shares the same architecture in both approaches. This network is an MLP comprised of three hidden layers, wherein the non-linearity is provided by the ELU activation function. More specifically, it is an MLP with the subsequent structure: $4 \to 100 \to 150 \to 100 \to 2$. It merits mention that this model's input is 4-dimensional, as the model's time variability is enabled by appending the time component t to the 2D data input $\boldsymbol{x}(t)$. The output is 2-dimensional, as it predicts the score at point $\boldsymbol{x}(t)$ at time t, i.e., $[s_1(\boldsymbol{\theta}), s_2(\boldsymbol{\theta})] = \boldsymbol{s_\theta} = \boldsymbol{s_\theta}(x_1(t), x_2(t), t, t)$. One could conjecture that a considerable enhancement of the model size would likely yield superior performance from the SA-DPM, albeit this would inevitably augment the already substantial training duration. Moreover, when considering more realistic scenarios involving high-dimensional data, the size of the network would inevitably be constrained due to memory-related limitations.

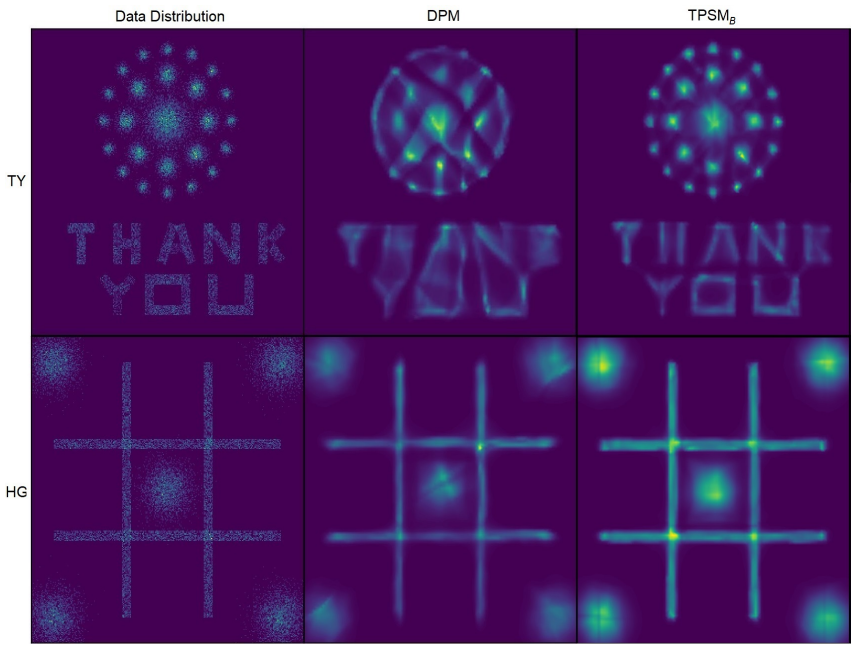

Figure 9: TPSM$_B$ visual results compared with SA-DPM.

Concerning standard image benchmark datasets, within the TPSM approach, we adopt the torch implementation (Wang, 2020) of the network originally proposed in (Ho et al., 2020) which, for the purpose of equitable comparison, is also utilized in SA-DPM. Similarly, within the DPSM approach, we opt for the even simpler basic U-Net, which originally presented the U-Net architecture in (Ronneberger et al., 2015). For all models, we implement a batch size of 32 on ImageNet and CelebA, and a batch size of 128 for CIFAR-10. In DPSM we adopt a learning rate of 10^{-3}, while in TPSM and SA-DPM this is reduced to 2×10^{-4}. Each neural network across all three methods was trained on a single GPU with RTX-2080 level of performance. As in the 2D data experiments, we utilize the ELU activation function.

The model implemented in the DPSM framework differs significantly, as it does not depend on time. In this instance, we elect to employ the initial U-Net as introduced and implemented in the pioneering U-Net paper (Ronneberger et al., 2015). The channel architecture is configured as follows: $(3, a, a*2, a*4, a*8, a*16, a*8, a*4, a*2, a, 3)$. When a is set to 64, this architecture aligns with the standard implementation. DPSM implementation necessitates interpolation techniques for precise likelihood estimation. For our experiments involving a thousand networks, we simply interpret the ODE as a piece-wise constant function, specifically, in CNF block i, the vector field is defined to be constant, and is described by the optimized score approximator $s_i(\boldsymbol{\theta})$ at time t_i. Utilizing the Euler method, likelihood estimation is performed with 5k steps and generation with 1k steps, avoiding interpolation in the generation process.

I Limitations and Future Work

I.1 Number of Computing Units and Likelihood Estimation in DPSM

The DPSM framework fundamentally relies on parallel computing, which necessitates a substantial number of computing units. In fact, the true advantages of parallelization within DPSM can only be fully exploited when one has access to computing clusters. Conversely, the TPSM framework can be effectively operated even by those with access to a single GPU, provided that the number of blocks is kept low. As demonstrated earlier, training two CNF blocks sequentially (as in the TPSM0 scenario) requires the same duration as training a single block within the SA-DPM framework. However, this approach results in enhanced performance, as evidenced in Table 2.

Though the training process can be performed in parallel, the generation and likelihood estimation processes are inherently sequential. In the context of DPSM, it is necessary to interpolate between successive models to carry out accurate likelihood estimation. The generation process, on the other hand, remains unaffected.

I.2 Tailored TPSM Models

In this study, due to computational simplifications, we chose to employ smaller, older models across all three frameworks (SA-DPM, TPSM, DPSM). Our main objective was to demonstrate that the latter two methods outperform the first one. In all instances, for fair comparison, we used models in the individual blocks (steps) in the TPSM (DPSM) approach that were equal to or smaller than the ones used in SA-DPM. Looking ahead, it would be valuable to remove these restrictions and attempt to develop custom models for the TPSM/DPSM frameworks. This would allow us to assess how much these strategies can enhance the state of the art.