ParaFlow: Parallel Sampling for Flow Matching Models

Jianrong Lu[1], Bangwei Li[1], Haomin Zhang[1], Zhuoya Gu[1], Yongqing Lu[2], Jianhai Chen[1](✉), and Qinming He[1](✉)

[1] Zhejiang University, Hangzhou, China
jrong.alvin@gmail.com
{chenjh919, hqm}@zju.edu.cn
[2] Guizhou Normal University, Guiyang, China

Abstract. This paper addresses the fundamental challenge of accelerating the inherently autoregressive sampling process in Flow Matching (FM) models through a numerical systems perspective. We introduce *ParaFlow*, a training-free framework that recasts sampling as a system of Triangular Nonlinear Equations (TNEs) to enable step-level parallelism. Unlike existing parallel samplers for diffusion models that require 2–3× more function evaluations (NFEs), we propose a velocity approximation scheme that leverages the temporal smoothness of FM trajectories. This allows ParaFlow to perform parallel sampling on a single GPU while requiring less total NFE count to the standard sequential sampler. We theoretically guarantee that our method converges to the exact autoregressive trajectory and that our approximation maintains negligible error bounds. Extensive experiments on Stable Diffusion 3 and Flux demonstrate that ParaFlow achieves up to **4.3**× wall-clock speedup with negligible impact on quality, making high-performance parallel sampling truly practical for resource-constrained settings. The source code is released in the supplementary materials. Code is available at `https://github.com/Jianrong-Lu/ParaFlow`.

1 Introduction

Flow Matching (FM) [5, 9, 16, 19, 20, 27, 40, 41, 50] has emerged as a leading framework for high-fidelity generative modeling, powering recent state-of-the-art systems [1,2,4,7,15,37,38,45–47] such as Stable Diffusion 3 [7] and Flux [1]. FM models learn a time-dependent vector field that transports a simple prior distribution to the data distribution via an ordinary differential equation (ODE). Thus, sampling reduces to numerically integrating this ODE, an inherently autoregressive procedure that requires tens of sequential evaluations of a large neural network, constituting a major bottleneck for real-time and interactive use.

Three main strategies have been explored to mitigate this cost. The first develops more efficient numerical solvers: high-order methods [8, 21, 22, 28, 49, 51] such as DPM-Solver [21,22,51] and task-specific solvers [8,28] reduce the number

of function evaluations (NFEs), enabling generation in fewer steps, but remain fully sequential. The second pursues model distillation [3,12,18,26], compressing multi-step teachers into student models capable of few-shot or even one-shot generation. Approaches such as Latent Consistency Models (LCMs) [24], based on Consistency Models [32], achieve strong results but require expensive retraining and degrade fidelity. Both methods reduce the step count but do not remove sequential dependencies. Recent attempts to introduce parallelism in generative models, such as ParaDiGMS [30] and ParaSolver [23], have primarily focused on the Diffusion Modeling paradigm. However, these methods face two major hurdles: (1) they are specifically tailored to the noise-prediction schedules of diffusion and do not directly translate to Flow Matching; and (2) they are computationally expensive, often requiring 2–3× more NFEs than sequential solvers to achieve convergence. Such high computational overhead frequently necessitates multi-GPU setups and high-bandwidth interconnects, making them impractical for standard single-GPU environments.

In this work, we propose ParaFlow, the first framework to achieve efficient parallel sampling specifically for Flow Matching models. Our approach departs from the high-overhead iterations of prior work by introducing a velocity approximation mechanism. By exploiting the observation that FM vector fields are generally smooth and exhibit near-linear dynamics, we use Newton's first-order divided difference to estimate future velocities within a parallel window.

This innovation allows ParaFlow to perform parallel sampling on a single GPU. Most importantly, by substituting expensive parallel model evaluations with lightweight approximations, ParaFlow achieves a total NFE count that is smaller to the sequential Euler baseline. This ensures that the speedup is derived purely from hardware-aware parallelism rather than increased computation.

Our main contributions are:

- We present the first formulation of parallel sampling for Flow Matching models by casting the ODE trajectory as a system of TNEs.
- We introduce a velocity approximation scheme that enables single-GPU parallel sampling, effectively removing the computational "tax" inherent in previous parallel samplers without quality loss.
- Through extensive validation on SD3 and Flux, we demonstrate wall-clock speedups of **1.4**–**4.3**× with theoretically guaranteed convergence to the exact autoregressive solution.

ParaFlow is a training-free, plug-and-play accelerator that is compatible with existing FM models, opening a new direction for efficient model inference.

2 Related Work

Accelerating generative model sampling is a long-standing research problem. For diffusion and flow-based models, which rely on iterative refinement, this is particularly critical.

Numerical ODE Solvers for Flow Matching. Flow Matching (FM) [5, 16,20] frames the sampling process as the integration of a learned time-dependent vector field. Early work [13,33–35,42] in score-based SDEs and their corresponding probability-flow ODEs enabled deterministic sampling and efficient likelihood evaluation. FM further refines this by training continuous normalizing flows [6] through conditional vector field regression along optimized probability paths, such as those [16, 17, 27, 39, 44] derived from diffusion or optimal transport, improving stability and speed with standard ODE solvers.

The choice of numerical integrator is crucial. First-order methods, such as Euler and DDIM [31], are computationally inexpensive but require a large number of steps to achieve high-quality results. To mitigate this problem, second-order solvers, such as Heun's method, are frequently employed [14]. For the most demanding tasks, higher-order solvers tailored specifically for diffusion models, such as the DPM-Solver family [21, 22, 51] and Bespoke Solvers [28, 29], offer improved sample quality with lower NFEs. Furthermore, methods like S4S [8], AYS [25] and DMN [48] have demonstrated potential for reducing discretization and global error. However, these methods still require strictly sequential execution, limiting opportunities for parallelization.

Parallel Sampling. ParaDiGMS [30] accelerates sampling by leveraging parallelized Picard iterations to simultaneously predict and iteratively refine future denoising steps. ParaTAA [36] reformulates the autoregressive diffusion sampling process as TNEs and uses fixed-point methods to solve multiple steps in parallel. ParaSolver [23] further models the sampling process as a system of banded nonlinear equations, and shows how to exploit structural properties and hierarchical initialization to preserve sample quality while greatly reducing inference time.

Our work distinguishes itself from these methods in two critical aspects. First, ParaDiGMS and ParaSolver are designed for Diffusion Models, whereas ParaFlow is the first to address the unique dynamics of FM. Second, existing parallel methods are computationally heavy, requiring multiple full-model evaluations per parallel iteration. This often requires distributing the workload across multiple GPUs to see real-world benefits. In contrast, ParaFlow's velocity approximation enables parallel execution on a **single GPU** with the **same NFE** as the sequential method, making it a significantly more practical solution for general users.

3 Preliminary: Flow Matching Models

Flow Matching models aim to learn a continuous-time process that transports a simple prior distribution p_0 (e.g., $\mathcal{N}(0, I)$) into a complex target data distribution p_1. This transformation is described by a time-dependent vector field $v_t : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$, parameterized by a neural network with parameters θ. For a sample trajectory $x_t, t \in [0, 1]$, the dynamics are governed by the ODE:

$$\frac{dx_t}{dt} = v_t(x_t, \theta), \quad x_0 \sim p_0. \tag{1}$$

The vector field v_t is learned by optimizing its underlying neural network parameters, such that the distribution of the terminal state x_1 approximates the target distribution p_1 when $x_0 \sim p_0$.

Sampling Process. To generate new samples, one first draws an initial point $x_0 \sim p_0$ and then integrates the ODE in Eq. (1) from $t = 0$ to $t = 1$. In practice, this integration is carried out numerically. A common approach is the first-order Euler method: given a discretization $0 = t_0 < t_1 < \cdots < t_N = 1$, the update rule is

$$x_{t_{i+1}} = x_{t_i} + (t_{i+1} - t_i) \cdot v(x_{t_i}, t_i, \theta), \quad i = 0, \ldots, N-1. \tag{2}$$

This discretization produces a sequence of intermediate states x_{t_i}. Since each update depends on the result of the previous step, the sampling procedure is inherently sequential, requiring N successive evaluations of the neural network.

4 Proposed Method: ParaFlow

4.1 Motivation

The autoregressive update in Eq. (2) constitutes the main bottleneck: each step depends on the previous one, precluding parallel computation and limiting sampling efficiency. We formalize this dependency as follows.

Definition 1 (Autoregressive Sampling Procedure). *Initiating with a sample $x_{t_0} \sim p_0$, the sampling process for an FM model using a numerical ODE solver is an autoregressive procedure of the form:*

$$x_{t_i} = x_{t_0} + \sum_{j=0}^{i-1} h_j \cdot v_j(x_{t_j}, t_j, \theta), \quad i \in \{1, \ldots, N\}, \tag{3}$$

where $h_j = t_{j+1} - t_j$ is the step size and v_j is the vector field evaluated at time t_j.

This formulation highlights the stepwise dependency. Viewing the entire trajectory $\{x_{t_0}, x_{t_1}, \ldots, x_{t_N}\}$ as unknown variables, this autoregressive process can be cast as a system of $N+1$ equations. Solving this system yields the full trajectory simultaneously, eliminating sequential dependencies and enabling parallel computation.

4.2 Recasting Autoregressive Sampling as Triangular Nonlinear Equation Solving

We can view the sequence of updates in Definition 1 as a system of TNEs. Let $\{\hat{x}_{t_0}, \ldots, \hat{x}_{t_N}\}$ be the unknown variables corresponding to the true sampling trajectory $\{x_{t_0}, \ldots, x_{t_N}\}$.

Definition 2 (TNEs for Flow Matching Sampling). *We define the system of TNEs for the autoregressive sampling procedure as:*

$$\mathcal{F}(\hat{x}_{t_0}, \ldots, \hat{x}_{t_N}) = \begin{cases} \hat{x}_{t_0} - x_{t_0} = 0, \\ \hat{x}_{t_i} - F_{i-1}(\hat{x}_{t_0}, \ldots, \hat{x}_{t_{i-1}}) = 0, \quad i \in \{1, \ldots, N\}, \end{cases} \tag{4}$$

where x_{t_0} is the initial sample from the prior and F_{i-1} is defined based on the ODE solver:

$$F_{i-1}(\hat{x}_{t_0}, \ldots, \hat{x}_{t_{i-1}}) = \hat{x}_{t_0} + \sum_{j=0}^{i-1} h_j \cdot v(\hat{x}_{t_j}, t_j, \theta). \tag{5}$$

This reformulation decouples the stepwise dependencies: given tentative states $\{\hat{x}_{t_j}\}$, the vector field $v(\hat{x}_{t_j}, t_j, \theta)$ can be evaluated for all timesteps $j \in \{0, \ldots, N-1\}$ in parallel. A crucial question then arises: does solving this system recover the same trajectory as the original sequential process?

Proposition 1 (Trajectory Equivalence (see supplementary for proof)). *The TNE system in* Eq. (4) *possesses a unique root that is identical to the sampling trajectory $\{x_{t_i}\}_{i=0}^{N}$ generated by the autoregressive procedure in* Eq. (3)*.*

This proposition ensures that by solving the TNEs, we can produce a sample indistinguishable in quality from that obtained via the standard autoregressive process.

4.3 Solving the TNE System with Fixed-Point Iteration

We can solve the system in Eq. (4) using various root-finding methods. Standard fixed-point iteration (FPI) is a natural choice. Given an initial guess for the entire trajectory $\{\hat{x}_{t_i}^{(0)}\}_{i=0}^{N}$, the update rule is:

$$\hat{x}_{t_i}^{(k+1)} = F_{i-1}(\hat{x}_{t_0}^{(k)}, \ldots, \hat{x}_{t_{i-1}}^{(k)}), \quad i \in \{1, \ldots, N\}, \tag{6}$$

with $\hat{x}_{t_0}^{(k+1)} = x_{t_0}$ for all k. Crucially, this update is step-level parallelizable: for any iteration k, the vector field evaluations $v(\hat{x}_{t_j}^{(k)}, t_j, \theta)$ across all $j \in \{0, \ldots, N-1\}$ can be computed simultaneously.

Proposition 2 (Convergence Guarantee (see supplementary for proof)). *Starting from any initial guess $\{\hat{x}_{t_i}^{(0)}\}_{i=0}^{N}$, the fixed-point iteration in* Eq. (6) *converges exactly to the autoregressive sampling trajectory $\{x_{t_i}\}_{i=0}^{N}$. This convergence is achieved in at most N iterations.*

The proof follows from the structure of the update mapping, which admits the autoregressive trajectory as its unique fixed point. In practice, the number of parallel iterations K required for convergence is typically much smaller than N, resulting in substantial acceleration.

4.4 Velocity Approximation via Newton's Divided Difference

While sampling can be performed in parallel across iterations, explicitly evaluating the vector field for all N velocities per iteration remains computationally prohibitive. In the standard Flow Matching framework, the formulation encourages straight-line probability paths, yielding trajectories that are generally smooth and locally near-linear. This characteristic suggests that velocity at future timesteps can be estimated.

However, we observe that a naive zero-order approximation, i.e., directly substituting future velocities with the most recently computed velocity, fails to capture temporal dynamics, leading to a marginal degradation in generation quality. To maintain high fidelity while reducing computation, we approximate the velocity changes via polynomial extrapolation using historical values. Specifically, we utilize Newton's first-order divided difference to approximate the future velocities.

Let $\{t_i, \ldots, t_{i+r-1}\}$ be the timesteps where the model is explicitly evaluated in parallel, and $\{t_{i+r}, \ldots, t_{i+N-1}\}$ be the timesteps where the velocity is extrapolated. Here, $1 \leq r \leq N$ denotes the number of velocities evaluated per parallel iteration. We perform the extrapolation utilizing t_{i+r-1} and t_{i+r-2} as historical references. Newton's first-order divided difference is defined as:

$$D_1 = v[t_{i+r-2}, t_{i+r-1}] = \frac{v_{t_{i+r-1}} - v_{t_{i+r-2}}}{t_{i+r-1} - t_{i+r-2}} \tag{7}$$

Using Newton's interpolation polynomial, our first-order estimate for the velocity field at a future timestep t_{i+j} for $j \in \{r, \ldots, p-1\}$ is given by:

$$v(\hat{x}_{t_{i+j}}, t_{i+j}, \theta) \approx v_{t_{i+r-1}} + D_1(t_{i+j} - t_{i+r-1}) \tag{8}$$

Note that setting $r = 1$ leads to identical NFEs per iteration as the traditional sequential sampling, making parallel sampling highly practical.

Proposition 3 (Estimation Error (see supplementary for proof)). *Let $\{x_{t_i}^{true}\}_{i=0}^{N}$ denote the trajectory obtained using the exact velocity field with the forward Euler method, and let $\{x_{t_i}^{approx}\}_{i=0}^{N}$ be the trajectory where velocity evaluations are approximated via Newton's first-order divided difference extrapolation. Under assumptions of spatial Lipschitz continuity and bounded time-derivatives, and assuming the total number of parallel iterations (which equals the total number of estimations) is K, the cumulative error in the final state after N steps is bounded by:*

$$e_N := \|x_{t_N}^{approx} - x_{t_N}^{true}\| = \mathcal{O}\left(\frac{K}{N^3}\right) \tag{9}$$

Remark. The theoretical bounds established in Proposition 3 mathematically guarantee that the error introduced by our velocity approximation is exceptionally small. Specifically, the global error decays at a cubic rate ($\mathcal{O}(1/N^3)$) for the first-order approximation with respect to the total number of sampling steps N. In standard Flow Matching inference, N is typically large (e.g.,

$N \in \{25, 50, 100\}$). Consequently, the polynomial denominator heavily dominates the numerator K where $K \leq N$ (the total number of parallel iterations/estimations). This indicates that even when aggressively parallelizing the process and approximating a vast majority of the trajectory, the cumulative deviation from the exact autoregressive solver remains strictly bounded and practically negligible.

Algorithm 1: ParaFlow: Parallel Sampling with Velocity Approximation

Input : Initial noise x_{t_0}, total steps N, timesteps $\{t_i\}_{i=0}^{N}$, parallel window size p, parallel model evaluation numbers r, error tolerance δ; maximum iterations K

Output: Generated sample $\hat{x}_{t_N}$.

Initialize $\{\hat{x}_{t_i}^{(0)} = x_{t_0}, i = 0, \ldots, p-1\}$.

$v_{t_{pre}} = v_{t_0}$; $t_{pre} = t_0$; $v_{t_0} \leftarrow v(x_{t_0}, t_0, \theta)$;

$i \leftarrow 1, k \leftarrow 0$;

while $i < N$ *and* $k < K$ **do**

`// 1. Parallel Model Evaluation`

Compute exact vector fields $v_{t_{i+j}} \leftarrow v(\hat{x}_{t_{i+j}}^{(k)}, t_{i+j}, \theta)$ in parallel for $j \in \{0, \ldots, r-1\}$.

`// 2. Velocity Approximation`

$v_{t_{cur}} = v_{t_{i+r-1}}$; $t_{cur} = t_{i+r-1}$;

$D_1 \leftarrow \frac{v_{t_{cur}} - v_{t_{pre}}}{t_{cur} - t_{pre}}$;

$v_{t_{i+j}} \leftarrow v_{t_{cur}} + D_1(t_{i+j} - t_{cur}), j \in \{r, \ldots, p-1\}$.

`// Update historical state`

$v_{t_{pre}} = v_{t_{cur}}$; $t_{pre} = t_{cur}$;

`// 3. Parallel State Update`

Update temporary states in parallel for $j \in \{0, \ldots, p-1\}$: $\hat{x}_{t_{i+j+1}}^{(k+1)} \leftarrow \hat{x}_{t_i}^{(k)} + \sum_{l=0}^{j}(t_{i+l+1} - t_{i+l}) \cdot v_{t_{i+l}}$;

`// 4. Convergence Check & Stride Calculation`

Compute error $e_j \leftarrow \frac{1}{D}\|\hat{x}'_{t_{i+j+1}} - \hat{x}_{t_{i+j+1}}\|^2$ for $j \in \{0, \ldots, p-1\}$

Find the smallest index s s.t. $e_s > \delta^2$; otherwise $s \leftarrow p$.

`// 5. Sliding Window Shift`

$i \leftarrow i + s, \quad k \leftarrow k + 1, \quad r \leftarrow \min(r, p), \quad p \leftarrow \min(p, N - i)$.

return $\hat{x}_{t_N}^{(K)}$;

4.5 Efficient Implementation with a Sliding Window

Solving for the entire trajectory of N steps in parallel can be memory-demanding. To make this practical, we implement the FPI within a sliding window of size $p \ll N$. We perform parallel iterations only on p subequations at a time. The window slides forward once the states within it have converged.

Initialization. We initialize all states within the first window to the starting noise vector: $\hat{x}_{t_i}^{(0)} = x_{t_0}$ for $i = 0, \ldots, p-1$. When the window slides, new states entering the window are initialized with the last converged state from the previous window.

Stopping Criterion. We define convergence within the window using a tolerance threshold δ. An iterate $\hat{x}_{t_i}^{(k)}$ is considered converged if the change from the previous iteration is small: $\frac{1}{D}\|\hat{x}_{t_i}^{(k)} - \hat{x}_{t_i}^{(k-1)}\|^2 \leq \delta^2$, where D is the number of pixels. The window slides forward by a stride, which is the number of contiguously converged states from the beginning of the window.

5 Experiment

We evaluate ParaFlow on two state-of-the-art text-to-image Flow Matching models: Stable Diffusion 3 (SD3) and Flux. Our primary goal is to quantify the trade-off between sampling speed and generation quality.

Experimental setup. All experiments were carried out on Ascend 910B GPUs. We evaluated generation quality using the Fréchet Inception Distance (FID) [11] and CLIP score [10], calculated over $5,000$ random prompts from the FluxPrompting dataset [43]. We report the average wall-clock time to generate a single image. Our baseline is the standard sequential Euler solver. We compare it with our proposed Euler + ParaFlow method across different total sampling steps ($N \in \{100, 75, 50, 25\}$). For ParaFlow, we use a default parallel window size of $p = 8$ and set the parallel model evaluation number $r = 1$.

We define the total steps (N) as the number of discrete steps in the sampling schedule. For a standard sequential method such as the Euler solver, the number of iterations (Iters) and the total number of function evaluations (NFE) are identical to N. For our parallel method, ParaFlow, Iters signifies the reduced number of sequential blocks executed, while the total computational workload, NFE, reflects the parallel execution and is calculated as Iters $\times\, r$.

5.1 Results on Stable Diffusion 3

We first evaluate ParaFlow on Stable Diffusion 3, generating 1024×1024 images from $5,000$ random prompts.

Performance across different step counts. Table 1 demonstrates that ParaFlow delivers consistent and significant wall-clock speedups while strictly preserving generation quality. Our most critical finding is the superiority of the velocity approximation scheme ($r = 1$). In the 100-step setting, ParaFlow with $r = 1$ achieves a **3.5× speedup** (5.20s vs. 18.22s) on a **single GPU**. Remarkably, this configuration outperforms the non-approximated parallel baseline ($r = 100$) which, despite utilizing 8 GPUs, only reaches a 3.3× speedup. This performance

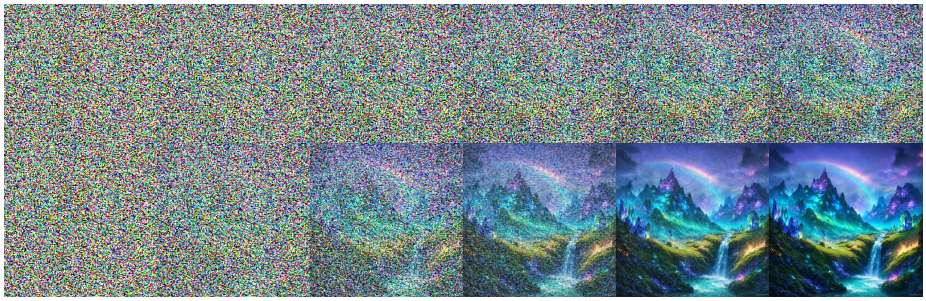
Table 1: Quantitative comparisons of different methods on Stable Diffusion 3 over $5,000$ random samples at 1024×1024. The visual comparisons are shown in the Appendix of the suplementary materials. The best results in each step are highlighted in **bold**. "↑" (resp. "↓") means the larger (resp. smaller), the better. Note that Euler + ParaFlow is evaluated with tolerance 0.005. The maximum iteration K is set to $29, 29, 24, 16$ for $N = 100, 75, 50, 25$ respectively. When $r > 1$, multi-GPU deployment is required to alleviate computational burden.

Steps	Method	Stable Diffusion 3					
		Iters↓	NFE↓	CLIP↑	FID↓	Time (s)↓	Speedup↑
100	Euler (1 GPU)	100.00	100.00	32.38	61.54	18.22	1.0×
	Euler + ParaFlow ($r = 100$, w/o estimate, 8 GPUs)	**26.47**	211.78	32.37	61.74	5.55	**3.3**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	28.34	**28.34**	32.17	61.96	**5.20**	3.5×
75	Euler (1 GPU)	75.00	75.00	32.37	62.14	13.57	1.0×
	Euler + ParaFlow (r=75, w/o estimate, 8 GPUs)	**22.96**	183.66	32.37	62.78	4.95	2.7×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	26.61	**26.61**	32.17	62.77	**4.81**	**2.8**×
50	Euler (1 GPU)	50.00	50.00	32.37	62.24	9.37	1.0×
	Euler + ParaFlow (r=50, w/o estimate, 8 GPUs)	**19.94**	159.49	32.36	62.79	4.43	**2.1**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	23.29	**23.29**	32.20	62.65	**4.42**	**2.1**×
25	Euler (1 GPU)	25.00	25.00	32.31	62.83	5.17	1.0×
	Euler + ParaFlow ($r = 25$, w/o estimate, 8 GPUs)	16.02	128.12	32.32	62.61	3.73	1.4×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	**15.97**	**15.97**	31.86	62.64	**3.23**	**1.6**×

gain is directly attributable to our elimination of the computational "tax" common in parallel solvers: whereas the $r = 100$ baseline requires an NFE of 211.78, our approximation achieves convergence with a significantly lower NFE of 28.34.

Furthermore, the quality metrics remain remarkably stable across all tested step counts. At $N = 100$, the FID increase is negligible (61.96 vs. 61.54), validating our theoretical guarantee that ParaFlow converges to the exact autoregressive trajectory. Even at low sampling budgets ($N = 25$), ParaFlow maintains a 1.6× speedup, proving that our framework is a robust, plug-and-play accelerator suitable for both high-fidelity and real-time generation tasks in resource-constrained, single-GPU environments.

Effect of tolerance. To further investigate the trade-off between speed and fidelity, we vary ParaFlow's tolerance parameter at 25 total steps (Table 2). Increasing the tolerance (e.g., 0.01) reduces the average number of iterations to 12.16, resulting in a 1.7× acceleration, at the cost of a modest increase in FID (55.59). Conversely, employing a stricter tolerance (e.g., 0.005) yields a FID of 54.00, closely aligning with the baseline Euler method (53.34), while still achieving a 1.4× speedup. This tunable parameter enables practitioners to adapt ParaFlow to task-specific requirements, providing a controllable trade-off between sample quality and inference efficiency. Representative visual comparisons are provided in Figure 2, which clearly illustrate the perceptual impact of different tolerance settings.



(a) iter = 0 **(b)** iter = 4 **(c)** iter = 8 **(d)** iter = 12 **(e)** iter = 16 **(f)** iter = 20

Fig. 1: All images are generated by the **stable-diffusion-3-medium** model with 50 sampling steps, using the prompt *"Create a dreamlike landscape featuring rolling hills, shimmering waterfalls, and towering mountains made entirely of crystal and gemstones that refract light into rainbows."* at resolution 1024×1024. The **top row** corresponds to the **Euler** method, while the **bottom row** corresponds to our proposed **Euler** + **ParaFlow** method (tolerance = 0.005).

Table 2: Quantitative comparisons on Stable Diffusion 3 with 25 sampling steps over $5,000$ random samples at 1024×1024. The best results are highlighted in **bold**. "↑" (resp. "↓") means the larger (resp. smaller), the better.

Method	Tolerance	Stable Diffusion 3					
		Iters↓	NFE↓	CLIP↑	FID↓	Time (s)↓	Speedup↑
Euler	0	25.00	25.00	32.31	63.83	5.17	1.0×
Euler + ParaFlow ($r = 25$, 8 GPUs)	0.01	**12.16**	97.32	32.31	65.61	**3.08**	**1.7**×
	0.007	13.96	111.72	32.31	64.63	3.38	1.5×
	0.005	16.02	128.12	32.32	64.11	3.73	1.4×

(a) standard **(b)** tolerance = 0.005 **(c)** tolerance = 0.007 **(d)** tolerance = 0.01

Fig. 2: The above images show results from the **stable-diffusion-3-medium** model at resolution 1024×1024 with 25 sampling steps. The prompt used is *"Abandoned church in night"*. The leftmost image corresponds to the **Euler** method, while the other three images correspond to our proposed **Euler** + **ParaFlow** method with parallel window size = 8 under different tolerance values (0.005, 0.007, 0.01).

Table 3: Quantitative comparisons of different methods on Flux over 5,000 random samples at 1024×1024. The visual comparisons are shown in the Appendix of the suplementary materials. The best results in each step are highlighted in **bold**. "↑" (resp. "↓") means the larger (resp. smaller), the better. Note that Euler + ParaFlow is evaluated with tolerance 0.01. The maximum iteration K is set to $25, 21, 18, 15$ for $N = 100, 75, 50, 25$ respectively. When $r > 1$, multi-GPU deployment is required to alleviate computational burden.

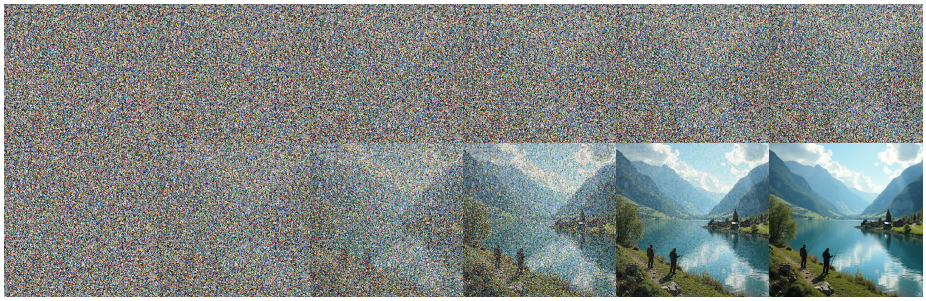
Steps	Method	Flux Model					
		Iters↓	NFE↓	CLIP↑	FID↓	Time (s)↓	Speedup↑
100	Euler (1 GPU)	100.00	100.00	32.31	61.47	87.94	1.0×
	Euler + ParaFlow ($r = 100$, w/o estimate, 8 GPUs)	23.49	187.95	32.40	61.45	22.16	**4.0**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	**22.93**	**22.93**	31.92	62.28	**20.45**	**4.3**×
75	Euler (1 GPU)	75.00	75.00	32.36	61.26	66.15	1.0×
	Euler + ParaFlow ($r = 75$, w/o estimate, 8 GPUs)	**19.32**	154.54	32.42	61.74	**18.38**	**3.6**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	20.90	**20.90**	31.92	61.76	18.47	**3.6**×
50	Euler (1 GPU)	50.00	50.00	32.38	61.36	44.38	1.0×
	Euler + ParaFlow ($r = 50$, w/o estimate, 8 GPUs)	**15.11**	120.86	32.45	61.85	**14.57**	**3.0**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	17.99	**17.99**	32.43	61.86	15.85	2.8×
25	Euler (1 GPU)	25.00	25.00	32.40	61.85	22.60	1.0×
	Euler + ParaFlow ($r = 25$, w/o estimate, 8 GPUs)	**12.79**	102.33	32.47	62.45	**12.42**	**1.8**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	14.93	**14.93**	32.42	62.48	13.29	1.7×

5.2 Results on Flux

We further validate ParaFlow on the Flux model, examining its performance across various sampling steps, resolutions, and window sizes.

Performance across different steps. We further validate ParaFlow on the Flux model, as shown in Table 3. Due to the massive parameter scale of Flux, the computational cost of sequential sampling is particularly high (e.g., 87.94s for $N = 100$). In this context, ParaFlow demonstrates even more significant efficiency gains than on smaller models. Our velocity approximation scheme ($r = 1, N = 100$) achieves a 4.3× wall-clock speedup (20.45s vs. 87.94s). This single-GPU configuration is faster than the 8-GPU non-approximated baseline ($r = 100$, 22.16s). This suggests ParaFlow successfully eliminates the computational "tax" (NFE overhead) that typically plagues parallel solvers. While the standard parallel approach requires 187.95 NFEs, our approximation converges with a mere 22.93 NFEs, a nearly 8× reduction in total computation.

Furthermore, ParaFlow maintains consistent acceleration across all step counts, achieving 3.6×, 2.8×, and 1.7× speedups for 75, 50, and 25 steps, respectively. The CLIP and FID scores for ParaFlow ($r = 1$) remain closely aligned with the sequential Euler baseline across the board (e.g., FID of 62.28 vs. 61.47 at $N = 100$). As shown in Figure 3, the perceptual details and semantic alignment are perfectly preserved. These results confirm that ParaFlow offers a transformative speed-to-fidelity ratio, making high-end parallel sampling accessible on standard hardware.



(a) iter = 0 **(b)** iter = 3 **(c)** iter = 6 **(d)** iter = 9 **(e)** iter = 12 **(f)** iter = 14

Fig. 3: All images are generated by the **FLUX.1-dev** model with 50 sampling steps, using the prompt *"A peaceful mountain village surrounded by rolling hills and a sparkling lake in the foreground, with fishermen casting their lines into the calm waters."* at resolution 1024×1024. The **top row** corresponds to the **Euler** method, while the **bottom row** corresponds to our proposed **Euler + ParaFlow** method (parallel window size = 8, tolerance = 0.01).

Table 4: Quantitative comparisons of Euler and Euler + ParaFlow on the Flux model over 5,000 random samples at 25 sampling steps. The best results in each resolution are highlighted in **bold** for Iters, Time, and Speedup only. "↑" (resp. "↓") means the larger (resp. smaller), the better. Note that Euler + ParaFlow is evaluated with tolerance 0.01 for 512×512 and 1024×1024, and 0.0028 for 2048×2048. When $r > 1$, multi-GPU deployment is required to alleviate computational burden.

Resolution	Method	Flux Model					
		Iters↓	NFE↓	CLIP↑	FID↓	Time (s)↓	Speedup↑
512×512	Euler (1 GPU)	25.00	25.00	32.19	59.97	11.31	1.0×
	Euler + ParaFlow ($r = 25$, w/o estimate, 8 GPUs)	**13.24**	105.90	32.23	60.31	4.81	2.3×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	16.00	**16.00**	32.21	60.14	**3.64**	**3.1**×
1024×1024	Euler (1 GPU)	25.00	25.00	32.40	61.85	22.60	1.0×
	Euler + ParaFlow ($r = 25$, w/o estimate, 8 GPUs)	**12.79**	102.33	32.47	62.45	**12.42**	**1.8**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	14.93	**14.93**	32.42	62.48	13.29	1.7×
2048×2048	Euler (1 GPU)	25.00	25.00	31.34	63.49	109.55	1.0×
	Euler + ParaFlow ($r = 25$, w/o estimate, 8 GPUs)	**14.30**	114.37	31.31	63.46	**65.86**	**1.7**×
	Euler + ParaFlow ($r = 1$, w/ estimate, 1 GPU)	17.20	17.20	32.23	63.59	78.25	1.4×

Effect of resolution. Table 4 demonstrates ParaFlow's scalability across different image resolutions with a 25-step schedule. Our method delivers consistent speedups: 2.3× at 512×512, 1.8× at 1024×1024, and 1.7× at 2048×2048. Crucially, the impact on FID and CLIP scores remains minimal across all resolutions, proving that ParaFlow's benefits generalize effectively to both low- and high-resolution synthesis. Complementary visual results are provided in Figure 4, which illustrate that image fidelity is preserved across scales while the generation time is substantially reduced.

(a) 512 × 512 **(b)** 1024 × 1024 **(c)** 2048 × 2048

Fig. 4: All images are generated by the **FLUX.1-dev** model with 25 sampling steps. The prompts used are: *"Skier gliding over fresh powder"* at resolution 512 × 512, *"A photographer focusing on the subject of a portrait, using natural light and composition to capture the perfect shot."* at resolution 1024 × 1024, and *"A charming farmhouse exterior with a white picket fence, a covered porch, and a garden full of colorful flowers and herbs."* at resolution 2048 × 2048. In each group, the **left image** corresponds to the **Euler** method, while the **right image** corresponds to our proposed **Euler + ParaFlow** method (parallel window size = 8, tolerance = 0.01, 0.01, and 0.0028 for the three resolutions, respectively).

Table 5: Quantitative comparisons of different window sizes for the FLUX model over 5,000 random samples at 512×512 with 50 sampling steps. The visual comparisons are shown in the Appendix of the suplementary materials. The best results are highlighted in **bold**. "↑" (resp. "↓") means the larger (resp. smaller), the better. Note that Euler + ParaFlow is evaluated with tolerance 0.01.

Method	Window Size	Flux Model					
		Iters↓	NFE↓	CLIP↑	FID↓	Time (s)↓	Speedup↑
Euler	1	50.00	50.00	32.13	59.96	23.80	1.0×
Euler + ParaFlow ($r = 50$, 8 GPUs)	32	14.76	472.44	32.17	59.83	15.12	1.6×
	24	**14.75**	354.02	32.17	59.83	12.21	1.9×
	16	14.83	237.27	32.17	59.90	9.18	2.6×
	8	15.44	123.51	32.16	59.82	**5.58**	**2.8**×

Effect of window size (p). We further analyze the influence of window size p on ParaFlow, using a 50-step schedule at 512 × 512 resolution (Table 5). The results reveal a clear trade-off: a smaller window size leads to a lower NFE and faster wall-clock time, at the cost of slightly more iterations. The window size of $p = 8$ provides the best balance, achieving a 2.8× speedup with almost identical image quality to the standard Euler solver. This flexibility allows practitioners to optimize ParaFlow for their specific hardware configurations. Figure 5 visually demonstrates this effect, showing that increasing the window size preserves fidelity while offering varying levels of acceleration.

Effect of parallel model evaluation numbers (r). Table. 6 demonstrates that decreasing the number of parallel model evaluations (r) significantly enhances sampling efficiency without compromising image quality. By lowering the evaluation count from 50 down to 1, the total Number of Function Evaluations (NFE) drops drastically from 472.44 to just 16.00. This substantial reduction in computational overhead maximizes hardware efficiency and accelerates inference times. Specif-

Table 6: Effect of parallel model evaluation numbers (r) for the FLUX model over $5,000$ random samples at 512×512 with 50 sampling steps. The visual comparisons are shown in the Appendix of the suplementary materials. The best results are highlighted in **bold**. "↑" (resp. "↓") means the larger (resp. smaller), the better. Note that Euler + ParaFlow is evaluated with tolerance 0.01, $p = 50$, and $K = 16$.

Method	Model Evaluation Numbers (r)	Flux Model					
		Iters↓	NFE↓	CLIP↑	FID↓	Time (s)↓	Speedup↑
Euler	50 (1 GPU)	50.00	50.00	32.13	59.96	23.80	1.0×
Euler + ParaFlow	50 (8 GPUs)	14.76	472.44	32.14	59.94	18.31	1.3×
	25 (8 GPUs)	14.32	368.02	32.15	59.97	12.53	1.9×
	8 (8 GPUs)	13.58	116.34	32.17	59.94	8.50	2.8×
	4 (4 GPUs)	15.34	64.98	32.16	59.95	7.93	3.0×
	1 (1 GPU)	16.00	**16.00**	32.14	59.96	**7.44**	**3.2**×

(a) standard **(b)** size = 8 **(c)** size = 16 **(d)** size = 24 **(e)** size = 32

Fig. 5: The above images show results from the **FLUX.1-dev** model at resolution 512×512 with 25 sampling steps. The prompt used is *"Library with lots of books on the floor"*. The leftmost image corresponds to the **Euler** method, while the other four images correspond to our proposed **Euler** + **ParaFlow** method under different parallel window sizes (8, 16, 24, 32) with tolerance fixed at 0.01.

ically, the speedup improves from 1.3× (taking 18.31 seconds on 8 GPUs for $r = 50$) to a peak of 3.2× (taking 7.44 seconds on a single GPU for $r = 1$) compared to the 23.80-second sequential baseline. Across all tested values of r, the generation fidelity remains exceptionally stable, yielding almost identical CLIP and FID scores. Ultimately, minimizing the evaluation count to $r = 1$ allows parallel sampling to achieve its highest wall-clock acceleration entirely on a single GPU.

5.3 Additional Qualitative Evaluation

We also provide qualitative results in Appendix of the suplementary materials, illustrating the images generated by ParaFlow on both Stable Diffusion 3 and Flux. The samples demonstrate that ParaFlow preserves high perceptual quality and semantic alignment while significantly reducing sampling time. Together with the quantitative analyses, these results confirm ParaFlow as a general and effective solution for fast and high-fidelity sampling.

6 Conclusion

This paper presents ParaFlow, the first parallel sampling framework for Flow Matching. By reformulating the ODE sampling process as a system of Nonlinear Equations (TNEs), we transform the autoregressive generation into a parallelizable root-finding problem. Our velocity approximation, based on Newton's divided difference, eliminates the typical computational overhead of parallel samplers, achieving significant speedups with fewer total NFEs than sequential baselines. Both theoretical analysis and empirical results confirm that ParaFlow converges to the exact trajectory with a cubic error decay, delivering up to 4.3× wall-clock speedup with negligible quality loss.

Acknowledgments and Disclosure of Funding. This work was supported by the Key R&D Program of Zhejiang Province under Grant No. 2025C01084 and the CAAI–MindSpore Open Fund, developed on the OpenI Community, under Contract No. CAAIXSJLJJ 2025 MindSpore 04.

References

1. Black Forest Labs: Introducing flux.1. https://www.blackforestlabs.ai/announcements/introducing-flux (August 2024), accessed: 2025-09-07
2. Cao, S., Chen, H., Chen, P., Cheng, Y., Cui, Y., Deng, X., Dong, Y., Gong, K., Gu, T., Gu, X., et al.: Hunyuanimage 3.0 technical report. arXiv preprint arXiv:2509.23951 (2025)
3. Chen, J., Xue, S., Zhao, Y., Yu, J., Paul, S., Chen, J., Cai, H., Han, S., Xie, E.: Sana-sprint: One-step diffusion with continuous-time consistency distillation (2025), https://arxiv.org/abs/2503.09641
4. Chen, J., Zhao, Y., Yu, J., Chu, R., Chen, J., Yang, S., Wang, X., Pan, Y., Zhou, D., Ling, H., et al.: Sana-video: Efficient video generation with block linear diffusion transformer (2025), https://arxiv.org/abs/2509.24695
5. Chen, R.T.Q., Lipman, Y.: Flow matching on general geometries. In: The Twelfth International Conference on Learning Representations (2024), https://openreview.net/forum?id=g7ohDlTITL
6. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. Advances in neural information processing systems **31** (2018)
7. Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., et al.: Scaling rectified flow transformers for high-resolution image synthesis. In: Forty-first international conference on machine learning (2024)
8. Frankel, E., Chen, S., Li, J., Koh, P.W., Ratliff, L.J., Oh, S.: S4s: Solving for a fast diffusion model solver. In: Forty-second International Conference on Machine Learning (2025), https://openreview.net/forum?id=9OaZCNbV2w
9. Geng, Z., Deng, M., Bai, X., Kolter, J.Z., He, K.: Mean flows for one-step generative modeling. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025), https://openreview.net/forum?id=uWj4s7rMnR
10. Hessel, J., Holtzman, A., Forbes, M., Bras, R.L., Choi, Y.: Clipscore: A reference-free evaluation metric for image captioning. arXiv preprint arXiv:2104.08718 (2021)

11. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)
12. Jiang, D., Liu, D., Wang, Z., Wu, Q., Jin, X., Liu, D., Li, Z., Wang, M., Gao, P., Yang, H.: Distribution matching distillation meets reinforcement learning. arXiv preprint arXiv:2511.13649 (2025)
13. Jo, J., Lee, S., Hwang, S.J.: Score-based generative modeling of graphs via the system of stochastic differential equations. In: International conference on machine learning. pp. 10362–10383. PMLR (2022)
14. Karras, T., Aittala, M., Aila, T., Laine, S.: Elucidating the design space of diffusion-based generative models. In: NeurIPS (2022)
15. Kong, W., Tian, Q., Zhang, Z., Min, R., Dai, Z., Zhou, J., Xiong, J., Li, X., Wu, B., Zhang, J., et al.: Hunyuanvideo: A systematic framework for large video generative models. arXiv preprint arXiv:2412.03603 (2024)
16. Lipman, Y., Chen, R.T., Ben-Hamu, H., Nickel, M., Le, M.: Flow matching for generative modeling. In: The Eleventh International Conference on Learning Representations (2022)
17. Lipman, Y., Havasi, M., Holderrieth, P., Shaul, N., Le, M., Karrer, B., Chen, R.T.Q., Lopez-Paz, D., Ben-Hamu, H., Gat, I.: Flow matching guide and code (2024), https://arxiv.org/abs/2412.06264
18. Liu, D., Gao, P., Liu, D., Du, R., Li, Z., Wu, Q., Jin, X., Cao, S., Zhang, S., Li, H., Hoi, S.: Decoupled dmd: Cfg augmentation as the spear, distribution matching as the shield. arXiv preprint arXiv:2511.22677 (2025)
19. Liu, J., Liu, G., Liang, J., Li, Y., Liu, J., Wang, X., Wan, P., ZHANG, D., Ouyang, W.: Flow-GRPO: Training flow matching models via online RL. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025), https://openreview.net/forum?id=oCBKGw5HNf
20. Liu, X., Gong, C., et al.: Flow straight and fast: Learning to generate and transfer data with rectified flow. In: The Eleventh International Conference on Learning Representations (2022)
21. Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., Zhu, J.: Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. Advances in Neural Information Processing Systems **35**, 5775–5787 (2022)
22. Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., Zhu, J.: Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. In: arXiv preprint arXiv:2211.01095 (2022)
23. Lu, J., Zhu, Z., Hou, J.: Parasolver: A hierarchical parallel integral solver for diffusion models. In: International Conference on Learning Representations (2025)
24. Luo, S., Tan, Y., Huang, L., Li, J., Zhao, H.: Latent consistency models: Synthesizing high-resolution images with few-step inference. arXiv preprint arXiv:2310.04378 (2023)
25. Sabour, A., Fidler, S., Kreis, K.: Align your steps: Optimizing sampling schedules in diffusion models (2024), https://arxiv.org/abs/2404.14507
26. Salimans, T., Ho, J.: Progressive distillation for fast sampling of diffusion models. In: International Conference on Learning Representations (2022)
27. Schusterbauer, J., Gui, M., Fundel, F., Ommer, B.: Diff2flow: Training flow matching models via diffusion model alignment. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 28347–28357 (2025)
28. Shaul, N., Perez, J., Chen, R.T.Q., Thabet, A., Pumarola, A., Lipman, Y.: Bespoke solvers for generative flow models (2023), https://arxiv.org/abs/2310.19075

29. Shaul, N., Singer, U., Chen, R.T.Q., Le, M., Thabet, A., Pumarola, A., Lipman, Y.: Bespoke non-stationary solvers for fast sampling of diffusion and flow models (2024), https://arxiv.org/abs/2403.01329
30. Shih, A., Belkhale, S., Ermon, S., Sadigh, D., Anari, N.: Parallel sampling of diffusion models. Advances in Neural Information Processing Systems **36** (2024)
31. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. In: International Conference on Learning Representations (2020)
32. Song, Y., Dhariwal, P., Chen, M., Sutskever, I.: Consistency models. International Conference on Machine Learning (2023)
33. Song, Y., Ermon, S.: Generative modeling by estimating gradients of the data distribution (2020), https://arxiv.org/abs/1907.05600
34. Song, Y., Ermon, S.: Improved techniques for training score-based generative models. Advances in neural information processing systems **33**, 12438–12448 (2020)
35. Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. In: International Conference on Learning Representations (2021)
36. Tang, Z., Tang, J., Luo, H., Wang, F., Chang, T.H.: Accelerating parallel sampling of diffusion models. In: Forty-first International Conference on Machine Learning (2024)
37. Team, T.H.: Hunyuanimage 2.1: An efficient diffusion model for high-resolution (2k) text-to-image generation. https://github.com/Tencent-Hunyuan/HunyuanImage-2.1 (2025)
38. Team, Z.I.: Z-image: An efficient image generation foundation model with single-stream diffusion transformer. arXiv preprint arXiv:2511.22699 (2025)
39. Tong, A., Fatras, K., Malkin, N., Huguet, G., Zhang, Y., Rector-Brooks, J., Wolf, G., Bengio, Y.: Improving and generalizing flow-based generative models with minibatch optimal transport. arXiv preprint arXiv:2302.00482 (2023)
40. Tong, A., FATRAS, K., Malkin, N., Huguet, G., Zhang, Y., Rector-Brooks, J., Wolf, G., Bengio, Y.: Improving and generalizing flow-based generative models with minibatch optimal transport. Transactions on Machine Learning Research (2024), https://openreview.net/forum?id=CD9Snc73AW, expert Certification
41. Tong, A.Y., Malkin, N., Fatras, K., Atanackovic, L., Zhang, Y., Huguet, G., Wolf, G., Bengio, Y.: Simulation-free Schrödinger bridges via score and flow matching. In: Dasgupta, S., Mandt, S., Li, Y. (eds.) Proceedings of The 27th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 238, pp. 1279–1287. PMLR (02–04 May 2024), https://proceedings.mlr.press/v238/tong24a.html
42. Vahdat, A., Kreis, K., Kautz, J.: Score-based generative modeling in latent space. Advances in neural information processing systems **34**, 11287–11302 (2021)
43. VincentGOURBIN: Fluxprompting. https://huggingface.co/datasets/VincentGOURBIN/FluxPrompting (2024)
44. Wang, S., Li, Z., Song, T., Li, X., Ge, T., Zheng, B., Wang, L., et al.: Differentiable solver search for fast diffusion sampling. arXiv preprint arXiv:2505.21114 (2025)
45. Wu, C., Li, J., Zhou, J., Lin, J., Gao, K., Yan, K., ming Yin, S., Bai, S., Xu, X., Chen, Y., Chen, Y., Tang, Z., Zhang, Z., Wang, Z., Yang, A., Yu, B., Cheng, C., Liu, D., Li, D., Zhang, H., Meng, H., Wei, H., Ni, J., Chen, K., Cao, K., Peng, L., Qu, L., Wu, M., Wang, P., Yu, S., Wen, T., Feng, W., Xu, X., Wang, Y., Zhang, Y., Zhu, Y., Wu, Y., Cai, Y., Liu, Z.: Qwen-image technical report (2025), https://arxiv.org/abs/2508.02324

46. Xie, E., Chen, J., Chen, J., Cai, H., Tang, H., Lin, Y., Zhang, Z., Li, M., Zhu, L., Lu, Y., Han, S.: Sana: Efficient high-resolution image synthesis with linear diffusion transformer (2024), https://arxiv.org/abs/2410.10629
47. Xie, E., Chen, J., Zhao, Y., Yu, J., Zhu, L., Lin, Y., Zhang, Z., Li, M., Chen, J., Cai, H., et al.: Sana 1.5: Efficient scaling of training-time and inference-time compute in linear diffusion transformer (2025), https://arxiv.org/abs/2501.18427
48. Xue, S., Liu, Z., Chen, F., Zhang, S., Hu, T., Xie, E., Li, Z.: Accelerating diffusion sampling with optimized time steps (2024), https://arxiv.org/abs/2402.17376
49. Zhao, W., Bai, L., Rao, Y., Zhou, J., Lu, J.: Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. Advances in Neural Information Processing Systems **36**, 49842–49869 (2023)
50. Zhao, W., Shi, M., Yu, X., Zhou, J., Lu, J.: Flowturbo: Towards real-time flow-based image generation with velocity refiner. Advances in Neural Information Processing Systems **37**, 4148–4176 (2024)
51. Zheng, K., Lu, C., Chen, J., Zhu, J.: Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. In: NeurIPS (2023)