WiSP-OSch: Solver Within-Step Parallelism and Order Scheduling for Diffusion Sampling

Víctor Lucas Rosada Canesin *
ENSEIRB-MATMECA, Bordeaux-INP
France
vrosadacane@enseirb-matmeca.fr

Julia Gusak
Inria, University of Bordeaux
France
yulia.gusak@inria.fr

Abstract

Diffusion sampling can be formulated as the integration of a continuous-time ODE, enabling the use of numerical solvers that efficiently reduce the number of denoising steps. Nevertheless, repeated neural network evaluations remain a major computational bottleneck. Most existing diffusion solvers rely on sequential function evaluations within each solver step and apply a fixed numerical integration method across time. We explore several solver modifications aimed at improving inference efficiency by relaxing these design choices. First, we study *within-step parallelism* (WiSP) by relaxing dependencies between function evaluations inside a single step of explicit Runge–Kutta solvers. By reusing a single shared state across these evaluations, this approach enables parallel execution within a step at the cost of reduced formal order. We include complementary analyses of parallel exponential solvers. Further, we observe that diffusion trajectories evolve unevenly over time and often contain extended regions with limited change. Motivated by this behavior, we propose *order scheduling* (OSch), which varies solver orders over time. Together, these findings indicate that carefully exploiting trajectory structure can reduce the computational cost of diffusion inference. We implemented all proposed methods[1] using the Diffusers[2] library API.

1 Introduction

Diffusion models have emerged as a powerful framework for generative modeling, with continuous-time formulations playing a central role in recent advances in efficient inference. In particular, the score-based framework (Song et al., 2021b) introduced stochastic and deterministic continuous-time dynamics whose terminal distributions correspond to the data distribution, enabling sampling to be framed as the numerical integration of differential equations. This perspective has motivated a large body of work on fast samplers that reduce the number of function evaluations (NFEs), ranging from DDIM (Song et al., 2021a) to higher-order approaches based on exponential integrators, most notably DPM-Solver (Lu et al., 2022) and its extensions. Despite these advances, inference remains computationally expensive, as each solver step is dominated by neural network evaluations, and most methods continue to rely on integration schemes in which function evaluations within each time step are performed sequentially and the solver structure is fixed across time.

A parallel line of work has explored the design of more sophisticated solvers for diffusion ODEs. DPM-Solver demonstrated that exponential integrators can substantially reduce NFEs by exploiting the semilinear structure of the diffusion ODE, and subsequent works such as DPM-Solver-v3 (Zheng et al., 2023) and DPM-Solver++ (Lu et al., 2023) further refined this approach by incorporating model-dependent statistics and adaptations for guided diffusion settings. Meanwhile, frameworks such as EDM (Karras et al., 2022) emphasized the importance of carefully chosen noise schedules for stable and efficient integration, while S^3 (Liu et al., 2024) proposed composing solvers to balance accuracy and computational cost. However, these methods largely retain classical solver

*The work was conducted during an internship at Inria, University of Bordeaux.

[1] https://github.com/vlcanesin/diffusers, https://github.com/vlcanesin/wisp-osch

[2] https://huggingface.co/diffusers

assumptions at the level of individual integration steps–such as sequential execution of intermediate function evaluations, reliance on locally accurate Taylor approximations and fixed solver order at each step–even in regimes where diffusion trajectories are integrated using extremely large step sizes. This mismatch between classical solver assumptions and diffusion inference practice has also been observed in recent solver search work, such as S4S (Frankel et al., 2025).

At the same time, several works have observed that diffusion trajectories exhibit substantial temporal redundancy, opening the door to alternative forms of efficiency. Feature reuse methods such as FRDiff (So et al., 2024) and DistriFusion (Li et al., 2024) exploit correlations across timesteps to amortize network evaluations and enable displaced patch parallelism respectively, while parallel-in-time approaches including ParaDiGMS (Shih et al., 2023) and SRDS (Selvam et al., 2024) relax strict step-to-step dependencies using Picard- and Parareal-style iterations respectively. While prior work has primarily focused on leveraging this redundancy for feature reuse or parallelization across time steps, a detailed analysis of how trajectory dynamics vary over time—and how this variation interacts with solver design choices at the level of individual integration steps—remains limited. In this work, we perform such a trajectory-level analysis and use it to motivate solver strategies that adapt to non-uniform temporal behavior.

Building on classical numerical methods, we investigate *within-step parallelism* (WiSP) by decoupling function evaluations within a single step of explicit Runge–Kutta solvers, enabling parallel execution at the cost of reduced formal order. We additionally include analyses of a fourth-order parallel exponential Runge–Kutta solver, adapted from (Luan, 2021), in Appendix D. We further introduce *order scheduling* (OSch), assigning solvers with different orders of accuracy to different regions of the trajectory based on observed dynamics, in contrast to fixed-order integration schemes. Together, our approach connects empirical trajectory analysis with solver design choices aimed at improving the efficiency of diffusion inference.

2 Proposed Methods

This section introduces the empirical analyses and solver modifications proposed throughout this work, with the goal of motivating the design choices evaluated in the Results section. Technical derivations, solver coefficients, and implementation details are deferred to the Appendix.

2.1 Trajectory Difference Analysis

To analyze temporal redundancy in diffusion trajectories, we compute the ℓ_1 norm of the difference between successive steps of the probability flow ODE averaged over spatial dimensions and batches. We perform this analysis across multiple dimensions: solver families (explicit Runge–Kutta solvers and exponential solvers of different orders), noise schedules (including EDM-style schedules), and model architectures. Across all configurations, we observe that the magnitude of these differences remains small relative to the magnitude of the solver states themselves. At the same time, the resulting difference curves exhibit clear non-uniformity, with extended regions of minimal variation alongside short intervals of more pronounced change. These observations motivate both the relaxation of strict solver assumptions enabled by trajectory stability and the heterogeneous solver designs explored in this work.

2.2 Within-Step Parallelism

Motivated by the goal of enabling parallel computation of intermediate function evaluations within a single solver step, we study within-step parallelism. Based on the observed small step-to-step changes in diffusion trajectories, we relax intra-stage dependencies for explicit Runge–Kutta solvers by zeroing coefficients associated with intermediate state updates, so that all stages reuse a shared state at the beginning of the step. While this modification sacrifices classical order guarantees, we observe empirically that the resulting solvers remain substantially more efficient than their sequential counterparts. To better understand this behavior, we provide in the Appendix C.3 a derivation of the local error, showing that these modified solvers preserve all time-derivative terms of $f(t, \boldsymbol{x}(t))$ up to the formal order of the original Runge–Kutta method, while the remaining error terms depend on derivatives with respect to $\boldsymbol{x}$. One plausible explanation is that, in diffusion ODEs, these state derivatives contribute less than the time-dependent terms in the regimes considered. The precise

definitions of all solvers considered are provided in Appendix C. A complementary adaptation to diffusion ODEs of a fourth-order exponential Runge–Kutta solver that supports partial parallelism among function evaluations within each step is described in Appendix D.

2.3 Order Scheduling

Finally, we introduce a simple and interpretable strategy for solver order scheduling. Based on the trajectory difference curves, we evenly divide the range of observed difference magnitudes into regions corresponding to the available solver orders. Thresholds determining order transitions are computed statically from these curves and applied uniformly across samples.

3 Results

3.1 Evolution of Trajectory Differences

Figure 3 reports the evolution of trajectory differences for three diffusion models trained on CIFAR-10: a lightweight U-Net, a standard Diffusers-style U-Net, and a DiT-S/4 model (see Appendix B). Across all models and schedules, the evolution of differences exhibits consistent structure. For a fixed timestep schedule, the difference curves are nearly identical across solvers, with only minor deviations for the EDM schedule between solvers that use different ODE formulations (see Appendix C.2). Two additional observations are that: the magnitude of the differences remains relatively small to the norm of the solver states themselves; the behaviour is similar across all models and solvers given the time schedule. Moreover, the evolution exhibits consistent non-uniform structure across models and solvers, particularly for tailored noise schedules such as EDM, which concentrate the changes in specific regions compared to linear schedules. The detailed description of solvers, schedules, and trajectory behavior is provided in Appendix A.

3.2 Within-Step Parallelism

We then evaluate within-step parallel Runge–Kutta (wisp-RK) variants that reduce inference cost by parallelizing computations within a solver step. These methods share the core idea of relaxing the dependencies between the stages of a step, thereby reducing the effective sequential depth.

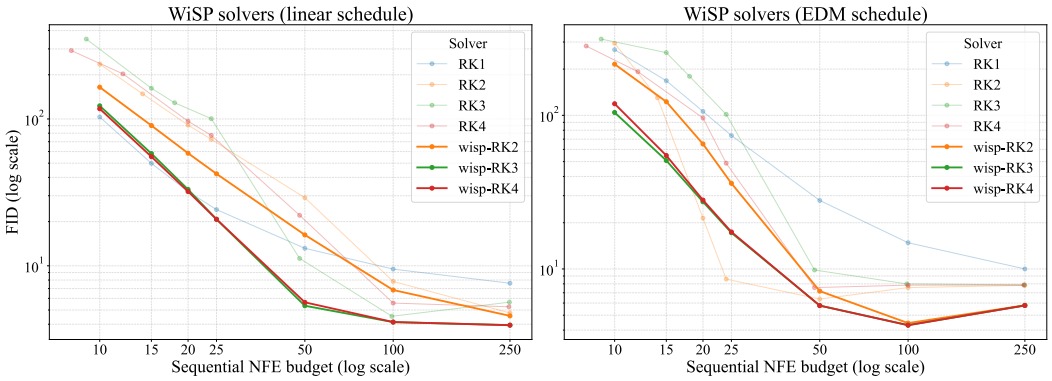


Figure 1: FID (↓) versus sequential number of function evaluations (sequential NFE) for explicit Runge–Kutta solvers of orders 1–4 and their within-step parallel variants (wisp-RK) under linear and EDM schedules.

Figure 1 shows that across most operating points, and for higher-order solvers, the within-step parallel variants outperform their sequential counterparts by achieving lower FID at a given NFE budget. An exception is observed for the second-order solver under the EDM schedule, where the sequential solver slightly outperforms its parallel variant. In addition to improved efficiency, the parallel solvers also achieve lower FID values for a fixed number of inference steps, indicating that the stage-parallel approximations do not degrade—and in many cases improve—sample quality. Additionally, in the Appendix D we investigate an adaptation of a fourth-order exponential Runge–Kutta

solver that supports partial parallelism among function evaluations within each step, and study its applicability to diffusion ODEs.

3.3 Order Scheduling

We now explore solver order scheduling in the context of a time-adapted timestep schedule. Figure 2 reports results obtained using a schedule, denoted *tEDM*, inspired by the EDM noise schedule but adapted to the variance preserving formulation (Song et al., 2021b), where integration is performed in time rather than over an SNR-based parameterization. Concretely, the power-law originally used to allocate noise levels in EDM is applied directly to the time variable, yielding a schedule with relatively finer resolution near the end of the trajectory and coarser resolution earlier on.

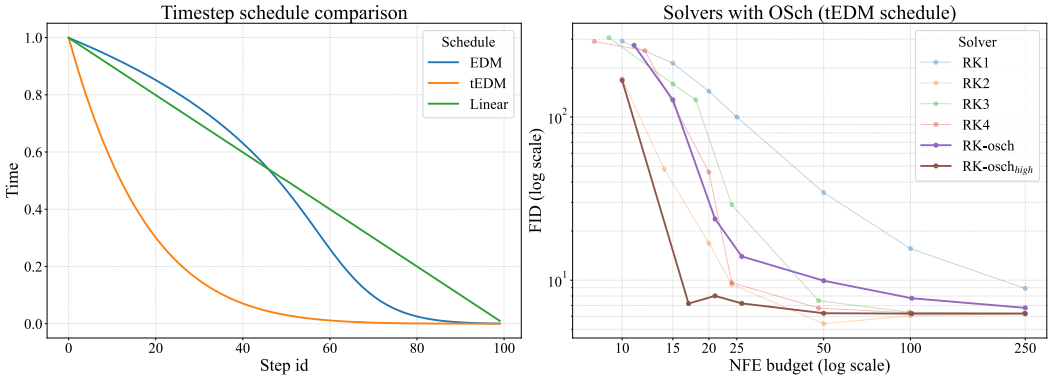


Figure 2: Left: timestep schedules (time vs step index) used for the RK solvers. Right: FID (↓) versus number of function evaluations for standard RK solvers, the composed RK solver with order scheduling based on trajectory differences, and a variant that preserves the highest-order solver during the early portion of the trajectory.

In addition to the baseline strategy described in Section 2.3, we consider the variant RK-osch$_{\text{high}}$ that preserves the highest-order solver (order four) during the early portion of the trajectory, up to the location of the difference peak (approximately the first 25% of steps), where the combination of larger timesteps and stronger trajectory variation benefits from higher-order integration. This choice is consistent with prior observations in diffusion sampling, notably those reported in PNDM (Liu et al., 2022), which highlight the sensitivity of the early reverse process to numerical errors. Intuitively, the learned diffusion vector field is reliable only in a narrow neighborhood around the true trajectory, and early deviations can push samples outside this region, which motivates the usage of more accurate, high-order solvers at the beginning of the trajectory. The results in Figure 2 support this interpretation: maintaining higher-order solvers early in the trajectory improves convergence, while switching to lower-order solvers later increases efficiency. Overall, the composed solvers reach competitive FID values within approximately 15–20 NFE, suggesting that trajectory-aware order scheduling is a promising direction for improving efficiency in moderate–to–low NFE regimes.

4 Discussion and Conclusion

Our empirical results suggest that, in the regimes considered in this work, diffusion ODE trajectories exhibit recurring patterns across different models, solvers, and noise schedules. Trajectory analysis enables practical strategies such as within-step parallelism, which reduces sequential depth in Runge–Kutta solvers, and order scheduling, which adjusts solver order based on expected variation. Importantly, within-step parallelism is orthogonal to existing parallelism approaches, operating at the solver level and complementing data- or tensor-level parallelism used to accelerate the model inference. These findings suggest that solver design can benefit from predictable patterns in the diffusion trajectory while remaining compatible with standard acceleration techniques. Future work should verify whether these observations hold for exponential solvers, more complex datasets, alternative generative models such as flow matching, and different pipelines, including latent diffusion.

ACKNOWLEDGMENTS

We thank Martin Parisot for helpful discussions. We also thank the members of the ELF Associate Team, supported by the Inria@SiliconValley program, for valuable exchanges. Experiments presented in this paper were carried out using the PlaFRIM[3] experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine. Victor was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

REFERENCES

Eric Frankel, Sitan Chen, Jerry Li, Pang Wei Koh, Lillian J Ratliff, and Sewoong Oh. S4S: Solving for a fast diffusion model solver. In *Forty-second International Conference on Machine Learning*, 2025.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 2022.

Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Kai Li, and Song Han. Distrifusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7183–7193, 2024.

Enshu Liu, Xuefei Ning, Huazhong Yang, and Yu Wang. A unified sampling framework for solver searching of diffusion probabilistic models. In *The Twelfth International Conference on Learning Representations*, 2024.

Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. *arXiv preprint arXiv:2202.09778*, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 2022.

Cheng Lu, Yuhao Zhou, et al. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2023.

Vu Thai Luan. Efficient exponential runge–kutta methods of high order: construction and implementation. *BIT Numerical Mathematics*, 61(2):535–560, 2021.

Connor Mooney, Zhongjian Wang, Jack Xin, and Yifeng Yu. Global well-posedness and convergence analysis of score-based generative models via sharp lipschitz estimates. *arXiv preprint arXiv:2405.16104*, 2024.

Nikil Selvam, Amil Merchant, and Stefano Ermon. Self-refining diffusion samplers: Enabling parallelization via parareal iterations. *Advances in Neural Information Processing Systems*, 37:5429–5453, 2024.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36:4263–4276, 2023.

Junhyuk So, Jungwon Lee, and Eunhyeok Park. Frdiff: Feature reuse for universal training-free acceleration of diffusion models. In *European Conference on Computer Vision*, pp. 328–344. Springer, 2024.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021a.

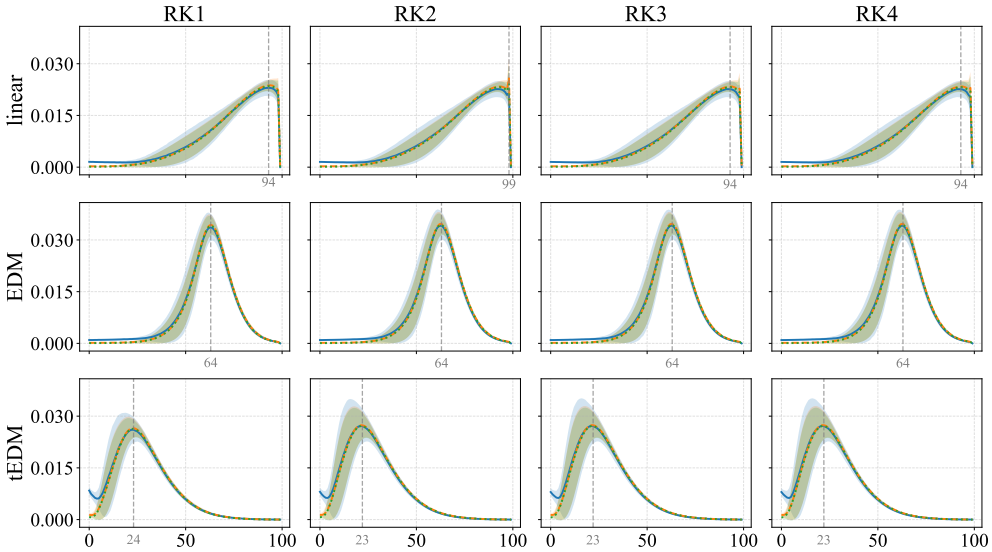
Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021b.

[3]https://www.plafrim.fr

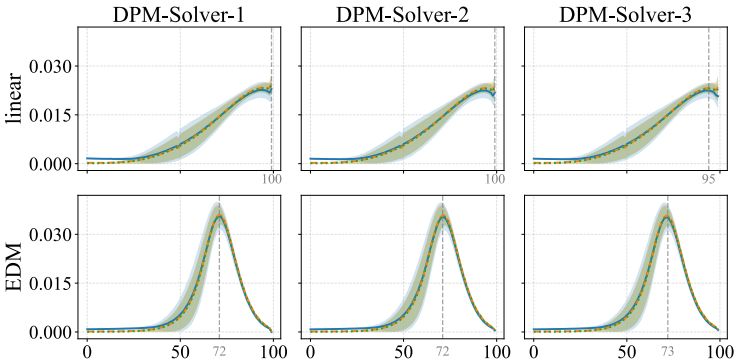
Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. *arXiv preprint arXiv:2310.13268*, 2023.

A Evolution of Differences

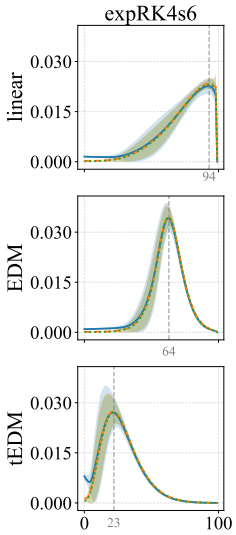
Figure 3 provides the full visualization of trajectory differences. The figure includes three diffusion models (lightweight U-Net, Diffusers-style U-Net, DiT-S/4) across multiple solvers and noise schedules, including linear, EDM, and tEDM schedules.



(a) Runge–Kutta solvers



(b) DPM-Solvers



(c) expRK4s6 solver

Figure 3: Evolution of trajectory differences across 100 denoising steps, averaged over 1024 images, for different models, solvers, and noise schedules. The blue, orange, and green curves correspond to the lightweight U-Net, the Diffusers-style U-Net, and the DiT-S/4 model, respectively. Shaded regions denote 99% confidence intervals and vertical markers indicate the timestep at which the maximum difference is observed.

B MODELS AND REPRODUCIBILITY

All experiments use unconditional pixel-level noise-prediction diffusion models trained on CIFAR-10 (32×32). This appendix briefly describes the models used and how to reproduce the results.

B.1 LIGHTWEIGHT U-NET

We use a lightweight U-Net obtained from the public Hugging Face repository `https://huggingface.co/karthik-2905/DiffusionPretrained`. The model follows a standard U-Net architecture with residual blocks, sinusoidal time embeddings, and a single attention block at the bottleneck.

This model was trained on CIFAR-10 and is used exclusively for trajectory difference analysis. It contains approximately *16.8M* parameters.

B.2 DIFFUSERS U-NET (DDPM CIFAR-10)

For all quantitative performance experiments, we use the U-Net released by Google and distributed via the Diffusers library: `https://huggingface.co/google/ddpm-cifar10-32`.

This is a standard DDPM U-Net with attention blocks and positional time embeddings, widely used as a benchmark model for CIFAR-10 diffusion. The model contains approximately *35.7M* parameters.

B.3 DIT-S/4

We additionally train and evaluate an unconditional pixel-level DiT-S/4 model, adapted from the official DiT implementation: `https://github.com/facebookresearch/DiT`. The model was trained on CIFAR-10 using 200k batches (batch size 256) with learning rate 10^{-4}, followed by 80k batches (batch size 256) with learning rate 10^{-5} for fine-tuning.

The DiT-S/4 model contains approximately *32.5M* parameters. It is used to assess whether trajectory difference behavior generalizes beyond convolutional architectures.

B.4 REPRODUCIBILITY

All results can be reproduced by loading the provided checkpoints (or training in the case of the DiT-S/4 model), running inference with the specified noise schedules and solvers, and computing trajectory differences or FID scores. Lastly, all FID points were computed from the denoising of 50k images.

C RUNGE–KUTTA SOLVERS AND ODE FORMULATIONS

This appendix summarizes the numerical solvers used throughout the paper, including classical Runge–Kutta (RK) methods and the parallelized variants introduced in this work. All solvers are written in a unified stage-based form to emphasize their structural similarities and differences. We additionally highlight the specific ODE formulations each solver is applied to, as differences in time parameterization and linear structure play an important role in the design and behavior of the methods discussed.

C.1 RK AND PRK SOLVERS

The key idea behind the parallel variants is simple: while the time locations of the internal stages are preserved, the intermediate state updates are removed. Concretely, the coefficients multiplying previous stages inside the arguments of $f(\cdot)$ are set to zero, allowing all stage evaluations to be computed independently. The final update then combines these stage evaluations using appropriately chosen weights. In what follows, we present the definitions of both the standard RK solvers and their parallel counterparts.

- Euler (RK1):

$$\begin{aligned} k_1 &= f(t_n, \boldsymbol{x}_n) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + hk_1 \end{aligned}$$

- Heun (RK2) and parallel version (wisp-RK2) respectively:

$$\begin{aligned} k_1 &= f(t_n, \boldsymbol{x}_n) \\ k_2 &= f\left(t_n + h, \boldsymbol{x}_n + hk_1\right) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{h}{2}(k_1 + k_2) \end{aligned} \qquad \begin{aligned} k_1 &= f(t_n, \boldsymbol{x}_n) \\ k_2 &= f\left(t_n + h, \boldsymbol{x}_n\right) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{h}{2}(k_1 + k_2) \end{aligned}$$

- RK3 and parallel version (wisp-RK3) respectively:

$$\begin{aligned} k_1 &= f(t_n, \boldsymbol{x}_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, \boldsymbol{x}_n + \frac{h}{2}k_1\right) \\ k_3 &= f\left(t_n + h, \boldsymbol{x}_n + h(-k_1 + 2k_2)\right) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{h}{6}(k_1 + 4k_2 + k_3) \end{aligned} \qquad \begin{aligned} k_1 &= f(t_n, \boldsymbol{x}_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, \boldsymbol{x}_n\right) \\ k_3 &= f\left(t_n + h, \boldsymbol{x}_n\right) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{h}{6}(k_1 + 4k_2 + k_3) \end{aligned}$$

- RK4 and parallel version (wisp-RK4) respectively:

$$\begin{aligned} k_1 &= f(t_n, \boldsymbol{x}_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, \boldsymbol{x}_n + \frac{h}{2}k_1\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, \boldsymbol{x}_n + \frac{h}{2}k_2\right) \\ k_4 &= f\left(t_n + h, \boldsymbol{x}_n + hk_3\right) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \qquad \begin{aligned} k_1 &= f(t_n, \boldsymbol{x}_n) \\ k_2 &= f\left(t_n + \frac{h}{3}, \boldsymbol{x}_n\right) \\ k_3 &= f\left(t_n + \frac{2h}{3}, \boldsymbol{x}_n\right) \\ k_4 &= f\left(t_n + h, \boldsymbol{x}_n\right) \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \frac{h}{8}(k_1 + 3k_2 + 3k_3 + k_4) \end{aligned}$$

C.2 Differences in the underlying ODE formulations

The RK and expRK solvers operate on the variance preserving (VP) formulation of the diffusion ODE (Song et al., 2021b):

$$\frac{d\boldsymbol{x}}{dt} = -\tfrac{1}{2}\beta(t)\left(\boldsymbol{x} - \frac{\hat{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}, t)}{\sqrt{1 - \bar{\alpha}(t)}}\right), \tag{1}$$

where $\beta(t)$ is the noise schedule, $\hat{\boldsymbol{\epsilon}}_\theta$ is the noise prediction network, and $\bar{\alpha}(t)$ denotes the cumulative noise level.

In contrast, DPM-Solver methods are derived from an alternative formulation:

$$\frac{d\boldsymbol{x}}{dt} = f(t)\boldsymbol{x} + \frac{g^2(t)}{2\sigma_t}\hat{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}, t), \tag{2}$$

with coefficients expressed explicitly in terms of α_t and σ_t, and where the signal-to-noise ratio α_t^2/σ_t^2 is strictly decreasing.

While both formulations describe equivalent diffusion processes in the continuous limit, they lead to fundamentally different numerical discretizations and error characteristics. For this reason, and to avoid conflating solver performance with modeling assumptions, we do not perform direct quantitative comparisons between VP-based Runge–Kutta solvers and DPM-based solvers in this paper.

C.3 Local Error Analysis of Parallel Runge–Kutta Solvers

We derive the Taylor expansion of the numerical updates produced by the parallel Runge–Kutta solvers and compare them to the exact solution of the underlying ODE. Throughout, we denote the state by $\boldsymbol{x}(t) \in \mathbb{R}^d$, corresponding to an image represented as a vector, and consider the ODE

$$\boldsymbol{x}'(t) = f(t, \boldsymbol{x}(t)). \tag{3}$$

We use the notation f_t and $f_{\boldsymbol{x}}$ to denote partial derivatives of f with respect to time and state, respectively.

Exact solution expansion. The exact solution admits the Taylor expansion

$$\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + h\boldsymbol{x}'(t) + \frac{h^2}{2}\boldsymbol{x}''(t) + \mathcal{O}(h^3) \tag{4}$$

$$= \boldsymbol{x}(t) + hf(t, \boldsymbol{x}) + \frac{h^2}{2}\Big(f_t(t, \boldsymbol{x}) + f_{\boldsymbol{x}}(t, \boldsymbol{x})f(t, \boldsymbol{x})\Big) + \mathcal{O}(h^3). \tag{5}$$

Second-order parallel solver. We now consider the second-order parallel Runge–Kutta solver, defined by the stages

$$k_1 = f(t_n, \boldsymbol{x}_n), \tag{6}$$

$$k_2 = f(t_n + h, \boldsymbol{x}_n), \tag{7}$$

and the update

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \frac{h}{2}(k_1 + k_2). \tag{8}$$

Expanding k_2 in time yields

$$k_2 = f(t_n, \boldsymbol{x}_n) + hf_t(t_n, \boldsymbol{x}_n) + \frac{h^2}{2}f_{tt}(t_n, \boldsymbol{x}_n) + \mathcal{O}(h^3). \tag{9}$$

Substituting into the update gives

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + hf(t_n, \boldsymbol{x}_n) + \frac{h^2}{2}f_t(t_n, \boldsymbol{x}_n) + \mathcal{O}(h^3). \tag{10}$$

Comparing equation 10 with the exact expansion equation 5, we observe that the numerical update exactly reproduces the time derivative term f_t, but omits the state-dependent contribution $f_{\boldsymbol{x}}f$. The resulting local truncation error is therefore

$$\varepsilon = \frac{h^2}{2}f_{\boldsymbol{x}}(t_n, \boldsymbol{x}_n)f(t_n, \boldsymbol{x}_n) + \mathcal{O}(h^3). \tag{11}$$

Higher-order parallel solvers. This behavior generalizes to higher-order parallel Runge–Kutta solvers. In fact, the update rules of wisp-RK3 and wisp-RK4 correspond to the Simpson and Simpson 3/8 quadrature rules respectively, and are both fourth-order accurate with respect to time.

Although the removal of intra-stage dependencies prevents the exact reproduction of terms involving derivatives with respect to $\boldsymbol{x}$, the solvers continue to exactly match all time derivatives of $f(t, \boldsymbol{x}(t))$ up to their nominal order. Concretely, for a parallel solver derived from a pth-order Runge–Kutta method, the numerical update reproduces all terms involving $f_t, f_{tt}, \ldots, f_{t^{p-1}}$, while neglecting mixed and state-dependent derivatives.

C.4 Missing State Derivatives

The remaining error terms involve derivatives of f with respect to the state variable $\boldsymbol{x}$. These terms are absent in the parallel formulation and constitute the dominant source of deviation from the classical solver.

We argue that, in the diffusion ODE setting, state derivatives of the score function are small relative to time derivatives for several complementary reasons. First, recent theoretical work establishes

the regularity of the score function: Mooney et al. (2024) show sharp Lipschitz bounds, indicating that small perturbations in the image $\boldsymbol{x}$ produce limited changes in $f(t, \boldsymbol{x})$. Second, both our trajectory analysis (Section 3.1) and the structure of the diffusion ODE suggest that the dynamics are primarily driven by the time-dependent component: consecutive states change slowly, the schedule-dependent coefficients dominate the updates, and the network outputs vary smoothly even under coarse timesteps. Taken together, these points motivate focusing on preserving time derivatives in parallel solvers. For diffusion ODEs, the error introduced by ignoring $f_{\boldsymbol{x}}$ is likely negligible in practice, particularly in the low-curvature, temporally redundant regions of the trajectory.

D Adapting the expRK4s6 Solver to Time-Dependent Linear Terms

In addition to explicit Runge–Kutta methods, we investigate a partially stage-parallel fourth-order exponential Runge–Kutta solver, expRK4s6 (Luan, 2021), originally proposed for semilinear partial differential equations. Although this solver follows a construction distinct from explicit wisp-RK methods, it shares a similar motivation: reducing effective sequential dependencies between stages while preserving high-order accuracy.

D.1 Problem Setting

The expRK4s6 solver is designed for problems of the form

$$u'(t) = Au(t) + g(t, u(t)), \tag{12}$$

where the linear operator A is assumed to be constant in time. Under this assumption, the method relies on an exact evaluation of the variation-of-constants formula, with exponential terms computed using a constant operator.

In the variance preserving (VP) formulation of the diffusion probability flow ODE, however, the corresponding linear term depends explicitly on time through the noise schedule (as shown in equation 1). This violates the core assumption of the original expRK4s6 construction. Treating the linear term as constant over a solver step therefore leads to an inexact integration of the linear component and introduces compounding local errors, particularly at coarse step sizes.

D.2 Quadrature-Based Adaptation

To address this issue, we modify the expRK4s6 solver by introducing explicit quadrature rules to approximate the integrals involving the time-dependent operator. Rather than freezing the linear term over each step, we approximate its contribution using numerical quadrature evaluated at intermediate time points.

We consider two quadrature schemes: a midpoint rule, which is second-order accurate; and a Simpson-type rule, which is fourth-order accurate. In the VP diffusion ODE, the time-dependent operator is linear in time. As a result, both the midpoint and Simpson quadrature schemes are exact for this component, ensuring that the modified solver preserves the formal order of accuracy of the underlying exponential integrator.

D.3 Toy Problem Validation

We validate the proposed quadrature-based adaptation on a one-dimensional test problem with a known closed-form solution:

$$u'(t) = \sin(t)\, u(t) + e^{-\cos(t)}, \qquad u(0) = e^{-1}. \tag{13}$$

This equation has the exact solution

$$u(t) = e^{-\cos(t)}. \tag{14}$$

Figure 4 compares the numerical accuracy of the original expRK4s6 solver, which ignores the time dependence of the linear term, with the midpoint- and Simpson-augmented variants. The results confirm that incorporating explicit quadrature substantially reduces integration error and restores

the expected convergence behavior. In particular, both quadrature schemes eliminate the dominant bias introduced by freezing the linear operator, with the Simpson variant achieving the lowest error across all tested step sizes.

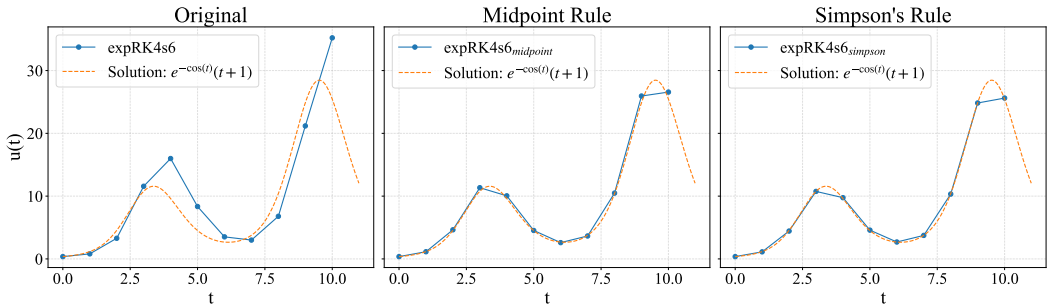


Figure 4: Accuracy of expRK4s6 variants on a one-dimensional test problem, $u'(t) = \sin(t)u(t) + e^{-\cos(t)}$, $u(0) = e^{-1}$. Incorporating midpoint or Simpson quadrature to account for the time-dependent linear term substantially improves accuracy compared to the original formulation, which assumes a constant operator.

D.4 Empirical Validation in Diffusion Sampling

Figure 5 reports FID versus the sequential NFE for the original expRK4s6 solver and its midpoint- and Simpson-augmented variants. Incorporating explicit quadrature for the time-dependent linear term significantly improves sample quality at a fixed NFE. Both quadrature variants are exact for the time-dependent linear term, eliminating the integration error introduced by treating it as constant and leading to consistent improvements in diffusion sampling performance.

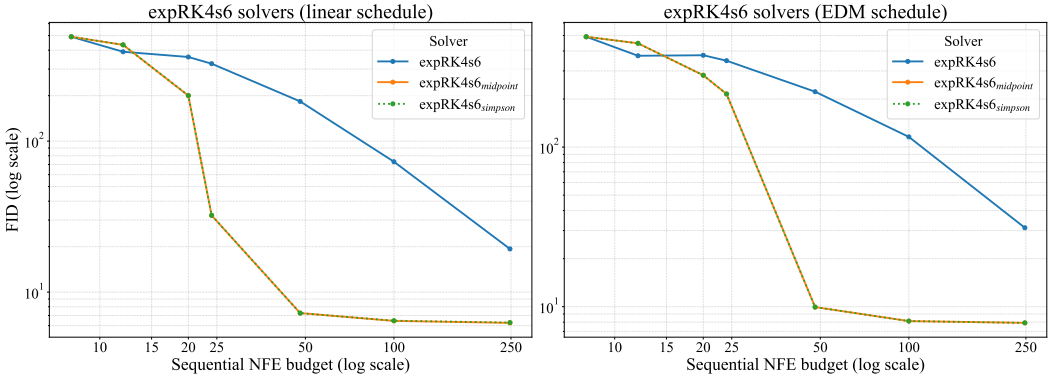


Figure 5: FID (↓) versus sequential NFE for expRK4s6 and its midpoint- and Simpson-augmented variants. Explicit quadrature for the time-dependent linear term improves performance across the operating points of interest.

D.5 Implications for Diffusion Solvers

While the adapted expRK4s6 solver is not fully stage-parallel, this study highlights an important design axis for diffusion solvers: selectively parallelizing or approximating subsets of solver stages while carefully preserving accuracy in the most sensitive components of the dynamics. The behavior and limitations observed here complement the wisp-RK results in the paper and suggest that further gains may be achievable by combining partial parallelism with structure-aware numerical integration.