Optimization Benchmark for Diffusion Models on Dynamical Systems

Fabian Schaipp
Inria, ENS, PSL Research University
Paris, France
fabian.schaipp@inria.fr

Abstract

In this work, we benchmark recent optimization algorithms for training a diffusion model for denoising flow trajectories. We observe that Muon and SOAP are highly efficient alternatives to AdamW (18% lower final loss). We also revisit several recent phenomena related to the training of models for text or image applications in the context of diffusion model training. This includes the impact of the learning-rate schedule on the training dynamics, and the performance gap between Adam and SGD.

1 Introduction

Over the last decade, the focus of optimization research has seen a shift towards applications in image classification and language modeling, particularly LLM pretraining. The training of *diffusion models*, despite their impressive success and wide range of applications, is usually absent from empirical validation in optimization research. Even the most extensive efforts on optimization benchmarking (Schmidt et al., 2021; Dahl et al., 2023; Kasimbeg et al., 2025) do not contain results on diffusion models. Further, it remains unclear whether newly proposed methods, such as SOAP (Vyas et al., 2025) or Muon (Jordan et al., 2024), are equally effective outside of LLM pretraining.

In this work, we validate whether recent trends in optimization for deep learning transfer to the training of diffusion models. In particular, our benchmark problem concerns training a diffusion model for denoising trajectories of dynamical systems, where the training data is obtained from fluid dynamics simulations. Our benchmark problem originally has been used for score-based data assimilation (Rozet & Louppe, 2023); compared to the setting of LLM pretraining, it is different in terms of model architecture and loss function, data domain and training regime (multi/single epoch).

In order to run multiple seeds and hyperparameter configurations for all methods, our computational constraints only allow for relatively small-scale problems ($\sim$ 23M parameters). Despite this limitation with respect to scale, the modeling technique from our benchmark problem has been successfully applied to diffusion-based data assimilation for regional and global weather and climate simulation (Manshausen et al., 2024; Schmidt et al., 2025; Andry et al., 2025). The findings of this benchmark might therefore be relevant and interesting to researchers who are training diffusion models for these scientific applications.

Benchmarking in optimization for machine learning. The most extensive optimization benchmarking effort in recent years has been the *AlgoPerf: Training Algorithms* benchmark (Dahl et al., 2023; Kasimbeg et al., 2025). It consists of a variety of workloads, such as image classification and reconstruction, speech recognition, language translation, molecular property prediction, and click-through rate prediction. With Shampoo (Gupta et al., 2018; Anil et al., 2020) being one of the competition winners, the benchmark sparked renewed interest in dense matrix preconditioning techniques and led to the development of new algorithms, such as SOAP (Vyas et al., 2025) and Muon

EurIPS 2025 Workshop on Principles of Generative Modeling (PriGM).

(Jordan et al., 2024). Semenov et al. (2025) and Wen et al. (2025) recently conducted extensive benchmarking for LLM pretraining. Here, we study whether these new methods can also shine for our diffusion training task. In particular, we compare the performance of SOAP, Muon and ScheduleFree (Defazio et al., 2024) to the baseline method AdamW (Loshchilov & Hutter, 2019).

Performance gap between Adam and SGD. In contrast to image classification with convolutional networks, where SGD and Adam perform equally well (if properly tuned), it is well known that SGD does not easily[1] achieve the same performance as Adam for language modeling tasks (Zhang et al., 2020; Kunstner et al., 2023). Kunstner et al. (2024) further showed that imbalance of the class labels is sufficient to observe a gap between Adam and SGD. It remains unclear in which way other factors (for example, components of the model architecture) can have the same effect. Here, we investigate whether SGD can close the gap to Adam for an instance of diffusion model training, where the argument of class imbalance is not applicable (as no class labels are involved).

Summary and main findings. Muon and SOAP prove to be highly efficient also for diffusion model training. Despite their higher runtime per step compared to AdamW, they achieve lower final loss values. ScheduleFree almost matches AdamW in terms of loss (without the need for scheduling), however we observe inferior generative quality. Similar effects can be observed for the wsd schedule, which leads us to the conjecture that the entire training trajectory (and not only final loss) is important for the quality of the trained diffusion model. We also observe a clear gap between Adam and SGD, which in this case can not be attributed to class imbalance.

2 Experimental Setup

Our experimental setup for training the diffusion model is following closely the setup of Rozet & Louppe (2023): they train a U-Net model (Ronneberger et al., 2015) which learns the score function of a dynamical system trajectory, obtained from the velocity field governed by the Navier-Stokes equations with Kolmogorov flow (Kochkov et al., 2021). Using the standard DDPM approach (Ho et al., 2020), the score function is learned by denoising data points sampled from the true distribution. We refer to Section A in the appendix for a detailed description of architecture and training data.

Hyperparameter tuning. For each optimizer, we tune learning rate and weight decay separately (see Fig. 8 for a detailed view on the grid search). In general, we run three different seeds for each setting, and average all metrics across seeds. If not specified otherwise, we run for 1024 epochs with a linear-decay learning-rate schedule. Compared to Rozet & Louppe (2023), we add warmup and gradient clipping by default (which lead to a minute reduction of the loss). A summary of the default hyperparameter settings is given in Table 1.

Computational cost. A single run over 1024 epochs with AdamW takes roughly one hour one a single NVIDIA A100 GPU (this includes the end-of-epoch evaluations). In total, we executed ~ 520 training runs. All experiments are conducted with Pytorch (Paszke et al., 2019) of version 2.5.1.

3 Results

Naming conventions. We use Adam and AdamW interchangeably. ScheduleFree always refers to the AdamW version presented by Defazio et al. (2024).

3.1 Main Benchmark

In this section, we compare the following optimizers:

- AdamW (Loshchilov & Hutter, 2019): Can be seen as the baseline method.
- Muon (Jordan et al., 2024): Designed for 2-dimensional weight matrices, and performs approximately steepest descent in the spectral norm (Bernstein & Newhouse, 2025). Muon has been

[1] Recent works show that SGD can close the gap to Adam also for language tasks when using very small batch sizes (Srećković et al., 2025; Marek et al., 2025), or when applying Adam only on the weights of the embedding layers (Zhao et al., 2025).

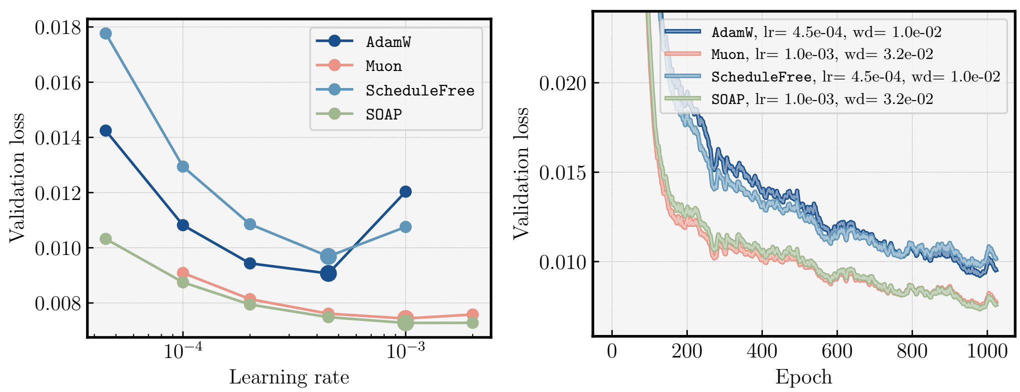


Figure 1: **(Left)** Final validation loss (averaged over the last five epochs) for each method and learning rate. Enlarged dot marks best learning rate. **(Right)** Validation loss curve for the best found setup for each method. Legend indicates learning rate (lr) and weight decay (wd) values. To obtain smoother curves we plot exponential moving averages with coefficient 0.95. See also Fig. 6.

reported to improve convergence speed of LLM pretraining compared to `AdamW` (Liu et al., 2025). See implementation details in Section B.1.

- `ScheduleFree` (Defazio et al., 2024): An adaptation of `AdamW` which does not require a learning-rate schedule (and therefore the length of training does not need to be pre-specified). We still use warmup, but afterwards the schedule is constant. `ScheduleFree` won the self-tuning track of the *AlgoPerf* benchmark (Kasimbeg et al., 2025).
- `SOAP` (Vyas et al., 2025): It combines the techniques from the `Shampoo` algorithm and `Adam`. `Shampoo` won the external tuning track of the *AlgoPerf* benchmark. As `SOAP` is a subsequent development and has been reported to perform better, we opt to run `SOAP` rather than `Shampoo`.

Pseudocode for all algorithms we compare can be found in Semenov et al. (2025, Appendix A).

Runtime per step. It is important to point out that `Muon` and `SOAP` have a larger runtime per step than the other methods. In our setup, the training time of one epoch is roughly $1.45\times$ larger for `Muon` and $1.72\times$ larger for `SOAP` (compared to `AdamW`). Given that runtime can significantly vary based on hardware and software setup, we focus on evaluation per steps, but also display loss curves with respect to training time. We use publicly available implementations for `Muon` and `SOAP` and do not perform any software optimization in order to speed up these two methods specifically for our task.

Main results. Fig. 1 shows the final validations loss for each learning rate and method (here we pick the best weight decay setting for each point). The best performing run for each method is displayed on the right. With respect to steps, `SOAP` achieves the best performance, closely followed by `Muon`. Over 1024 epochs (equal to 26.6K steps), `Muon` and `SOAP` achieve a loss value that is 18% lower than the final loss of `AdamW`. `ScheduleFree` improves over `AdamW` early on in training, but falls slightly short in the end. With respect to runtime (see Fig. 7), `Muon` performs best; `SOAP` converges equally fast as `AdamW`, but reaches a lower final loss. We stress that **these results might vary** based on hardware setup and software optimization.

What happens if we simply train `AdamW` for longer? When comparing in terms of runtime, the advantage of `Muon` and `SOAP` over `AdamW` is reduced significantly. This leads to the question whether `AdamW` can match the final loss of `SOAP`/`Muon` if we simply train for more epochs. Fig. 2 shows that this is not the case. In this sense, `SOAP` and `Muon` achieve lower final loss values even with the same (or lower) runtime budget. Note that for the `AdamW` runs over 2048 epochs, we re-tune the learning rate in order to account for the extended training length, but keep weight decay fixed to 10^{-2}. As sensitivity to weight decay is generally rather small, we do not expect this to impact the conclusion.

Impact of schedule on generative quality. A major drawback of the linear-decay (or `cosine`) schedule is that the entire schedule depends on training length, which in consequence needs to be specified ahead-of-time. As an alternative, the `wsd` schedule ("warmup-stable-decay") has been

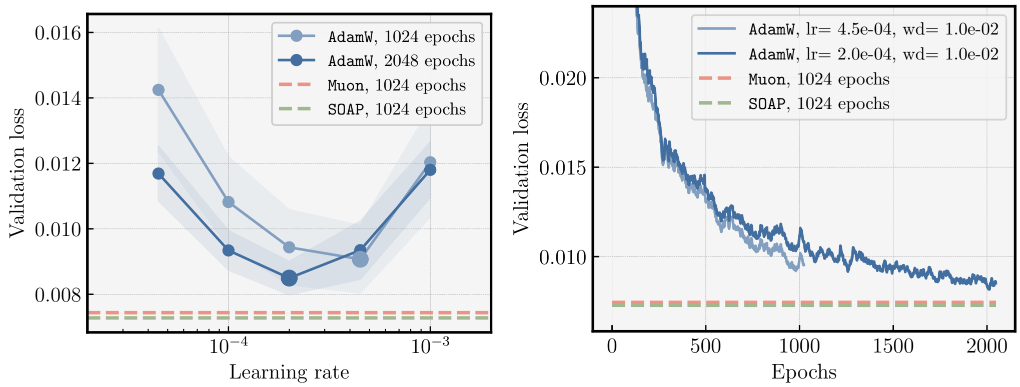


Figure 2: **(Left)** Final validation loss (averaged over the last five epochs, with band of one standard deviation over three seeds). Horizontal line marks best final loss for `Muon` and `SOAP` after 1024 epochs. **(Right)** Validation loss curve for the best `AdamW` run over 1024 and 2048 epochs (smoothened by exponential moving averages with coefficient 0.95).

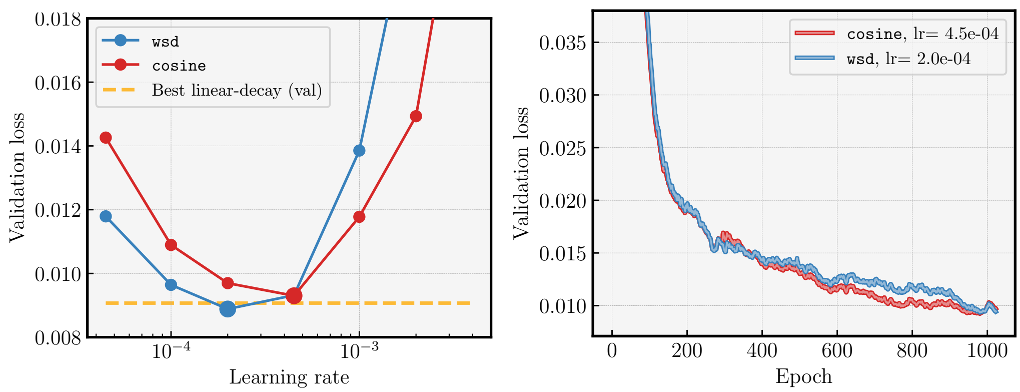
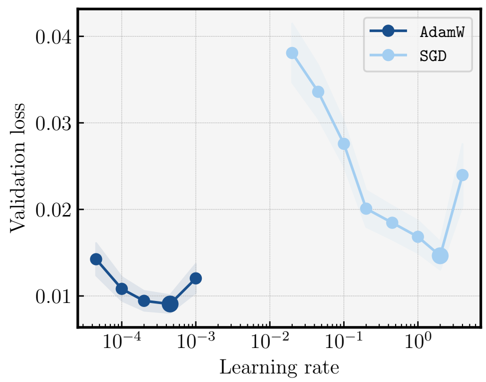


Figure 3: **(Left)** Final validation loss (averaged over the last five epochs) for `wsd` (with a cooldown length of 20%) and `cosine` schedule across peak learning rate. Enlarged dot marks best learning rate. **(Right)** Validation loss curve for the best found setup for each schedule (smoothened by exponential moving averages with coefficient 0.95).

proposed in the context of LLM pretraining: it keeps the learning rate constant, and a linear cooldown can be performed at any time (Hu et al., 2024; Hägele et al., 2024). The `wsd` schedule matches or surpasses the performance of `cosine` for LLM pretraining (Hägele et al., 2024). Here, we find that, in terms of loss values, the same is true for the diffusion model training we consider (see Fig. 3). Similar to empirical and theoretical findings by Hägele et al. (2024); Schaipp et al. (2025), the optimal peak learning rate for `wsd` is roughly half of the optimal one for `cosine`. However, it seems that **generative quality becomes less stable when using the `wsd` schedule** (see Fig. 9); for the learning rate with minimal loss, the generated trajectories are of lower quality.

Mismatch of loss value and generative quality for `ScheduleFree`. We find that specifically for `ScheduleFree`, similar loss values do not correspond to similar quality of generated trajectories (see Figs. 5 and 10). We conjecture that this is partially due to the missing learning-rate annealing: using a `wsd` schedule for `ScheduleFree` improves generative quality, at least for some hyperparameter configurations (Fig. 11).

Practical takeaways. The optimal learning rate for `Muon` and `SOAP` is roughly twice as large as the optimal learning rate for `AdamW`. We are confident that this is not problem-specific, as the same has been found by Semenov et al. (2025) for LLM training. For our problem, sensitivity to the weight decay value is much smaller than to learning rate (see Fig. 8). Overall, `SOAP` is the method that is least sensitive to learning rate/weight decay.



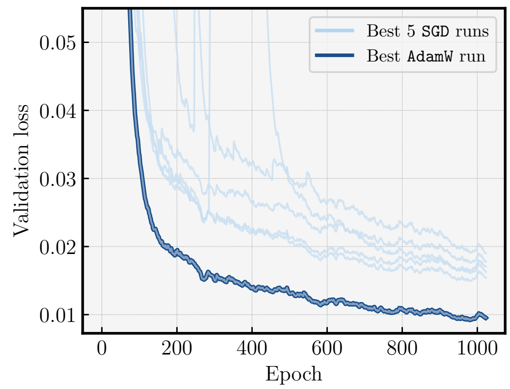


Figure 4: **(Left)** Final validation loss (averaged over the last five epochs) for each method and learning rate. **(Right)** Validation loss curve for the best found `AdamW` setup, and the best five `SGD` setups (smoothened by exponential moving averages with coefficient 0.95).

3.2 Gap Between `AdamW` and `SGD`

Here, we investigate whether there is a significant gap in training/validation loss between `AdamW` and `SGD`, when both methods are well-tuned. Starting from Kunstner et al. (2023), this gap and its possible reasons have been studied extensively, mainly for image and language tasks. Our setup will add another perspective, as we study a different training task (diffusion), and data type (from turbulence simulation rather than images or text); in particular, the explanation that class imbalance causes the gap between `AdamW` and `SGD` can not be applied here, as there are no class labels involved.

Fig. 4 shows a significant gap in terms of validation loss between `AdamW` and `SGD`. For training loss, the results are qualitatively the same (plots not shown). The visual quality of the generated trajectories trained with `SGD` are also clearly inferior, despite extensive hyperparameter tuning (see Fig. 12). This leads us to the conclusion that for this problem instance other factors must be at play that explain the gap between `Adam` and `SGD`. We leave it for future work to investigate what these factors could be in the context of diffusion (for example, the role of the model architecture).

4 Conclusion

We show that `Muon` and `SOAP` are convincing alternatives to `AdamW` for training of diffusion models. Despite a larger runtime per step, they reach significantly lower loss values than `AdamW`; moreover, their advantage remains even when compared to `AdamW` with twice the epoch budget. Further, for our problem, the choice of learning-rate schedule, or using `ScheduleFree`, can hurt the generative quality of the model, even though the same loss value is achieved. We therefore conclude with the hypothesis that the entire optimization trajectory might be important for the generative quality of the model.

We hope that the results of this benchmark can inspire future work to understand the phenomena observed here: What is the reason for the superior performance of `Muon` and `SOAP`? What causes the gap between `SGD` and `Adam`? How is the (final) generative quality affected by the learning dynamics for diffusion models?

Acknowledgments and Disclosure of Funding

Fabian Schaipp is supported by the French government under the management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), and the European Research Council Starting Grant DYNASTY – 101039676. This work was granted access to the HPC resources of IDRIS under the allocation 2025-AD011016024 made by GENCI.

References

Gérôme Andry, François Rozet, Sacha Lewin, Omer Rochman, Victor Mangeleer, Matthias Pirlet, Elise Faulx, Marilaure Grégoire, and Gilles Louppe. Appa: Bending weather dynamics with latent diffusion models for global data assimilation. arXiv:2504.18720, 2025.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. arXiv:2002.09018, 2020.

Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. In *International Conference on Machine Learning*, 2025.

Alberto Carrassi, Marc Bocquet, Laurent Bertino, and Geir Evensen. Data assimilation in the geosciences: An overview of methods, issues, and perspectives. *WIREs Climate Change*, 9(5): e535, 2018.

George E. Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, Juhan Bae, Justin Gilmer, Abel L. Peirson, Bilal Khan, Rohan Anil, Mike Rabbat, Shankar Krishnan, Daniel Snider, Ehsan Amid, Kongtao Chen, Chris J. Maddison, Rakshith Vasudev, Michal Badura, Ankush Garg, and Peter Mattson. Benchmarking neural network training algorithms. arXiv:2306.07179, 2023.

Aaron Defazio, Xingyu Yang, Ahmed Khaled, Konstantin Mishchenko, Harsh Mehta, and Ashok Cutkosky. The road less scheduled. In *Advances in Neural Information Processing Systems*, volume 37, pp. 9974–10007, 2024.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, volume 80, pp. 1842–1850, 2018.

Alex Hägele, Elie Bakouch, Atli Kosson, Loubna Ben allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In *Advances in Neural Information Processing Systems*, volume 37, pp. 76232–76264, 2024.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Shengding Hu, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Xinrong Zhang, Zhen Leng Thai, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *First Conference on Language Modeling*, 2024.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. Blog post.

Priya Kasimbeg, Frank Schneider, Runa Eschenhagen, Juhan Bae, Chandramouli Shama Sastry, Mark Saroufim, Boyuan Feng, Less Wright, Edward Z. Yang, Zachary Nado, Sourabh Medapati, Philipp Hennig, Michael Rabbat, and George E. Dahl. Accelerating neural network training: An analysis of the algoperf competition. In *International Conference on Learning Representations*, 2025.

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between SGD and Adam on transformers, but sign descent might be. In *International Conference on Learning Representations*, 2023.

Frederik Kunstner, Alan Milligan, Robin Yadav, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why Adam outperforms gradient descent on language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30106–30148, 2024.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training. arXiv:2502.16982, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Peter Manshausen, Yair Cohen, Peter Harrington, Jaideep Pathak, Mike Pritchard, Piyush Garg, Morteza Mardani, Karthik Kashinath, Simon Byrne, and Noah Brenowitz. Generative data assimilation of sparse weather station observations at kilometer scales. arXiv:2406.16947, 2024.

Martin Marek, Sanae Lotfi, Aditya Somasundaram, Andrew Gordon Wilson, and Micah Goldblum. Small batch size training for language models: When vanilla SGD works, and why gradient accumulation is wasteful. arXiv:2507.07101, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035. 2019.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

François Rozet and Gilles Louppe. Score-based data assimilation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 40521–40541, 2023.

Fabian Schaipp, Alexander Hägele, Adrien Taylor, Umut Simsekli, and Francis Bach. The surprising agreement between convex optimization theory and learning-rate scheduling for large model training. In *International Conference on Machine Learning*, volume 267, pp. 53267–53294, 2025.

Jonathan Schmidt, Luca Schmidt, Felix M. Strnad, Nicole Ludwig, and Philipp Hennig. A generative framework for probabilistic, spatiotemporally coherent downscaling of climate simulation. *npj Climate and Atmospheric Science*, 8(1), July 2025.

Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley - benchmarking deep learning optimizers. In *International Conference on Machine Learning*, volume 139, pp. 9367–9376, 2021.

Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. Benchmarking optimizers for large language model pretraining. arXiv:2509.01440, 2025.

Teodora Srećković, Jonas Geiping, and Antonio Orvieto. Is your batch size the problem? revisiting the Adam-SGD gap in language modeling. arXiv:2506.12543, 2025.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. SOAP: improving and stabilizing Shampoo using Adam for language modeling. In *International Conference on Learning Representations*, 2025.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. arXiv:2509.02046, 2025.

Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank Reddi, Sanjiv Kumar, and Suvrit Sra. Why are adaptive methods good for attention models? In *Advances in Neural Information Processing Systems*, volume 33, pp. 15383–15393, 2020.

Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham M. Kakade. Deconstructing what makes a good optimizer for autoregressive language models. In *International Conference on Learning Representations*, 2025.

A Overview of Model and Dataset

Background on the learning task. Data assimilation is a central problem in many scientific domains that involve noisy measurements of complex dynamical systems, such as oceans or atmospheres (see Carrassi et al. (2018); Rozet & Louppe (2023) and references therein). Data assimilation can be seen as an inverse problem: the task is to estimate the distribution of true trajectories of the dynamical system, given a noisy measurement. The main contribution of Rozet & Louppe (2023) is to estimate this distribution based on a learned score function of true trajectories. This score function is obtained via standard diffusion model training. One advantage of their approach is that training and estimation can be performed entirely decoupled. For our purpose of studying the performance of optimization algorithms, we focus solely on the training task.

Dataset. Our data generation procedure is identical to Rozet & Louppe (2023). For the sake of completeness, we describe the main steps below. The input data for the diffusion model are snapshots of the (2-dimensional) velocity field which is governed by the Navier-Stokes equations. We follow Rozet & Louppe (2023); Kochkov et al. (2021) by solving the Navier-Stokes equations on a two-dimensional domain $[0, 2\pi]^2$, with periodic boundary conditions, a large Reynolds number $Re = 1000$, a constant density, and an external forcing corresponding to Kolmogorov forcing with linear damping (cf. Kochkov et al. (2021)). The data is generated by solving 1024 independent trajectories of the Navier-Stokes equations (using `jax-cfd`) on a grid of resolution 256×256. Each trajectory consists of 128 snapshots, which are then down-sampled to a resolution of 64×64, and filtered on the second half of the trajectory. We split the 1024 trajectories into training (80%), validation (10%) and test (10%) set.

During training, for each trajectory in the batch a random window of five snapshots is sampled with random starting point; this leaves us with input data of the shape $(b, 10, 64, 64)$, where b is the batch size.

Model architecture. The model is a `U-Net` architecture (Ronneberger et al., 2015) with three hidden convolutional layers, of channel sizes $(96, 192, 384)$. Further, we use a time embedding dimension of 64. The model has 22.9 million trainable parameters. For more details, we refer to Rozet & Louppe (2023).

B Supplementary Material on Experiments

Our code and all training logs of this benchmark are publicly available at `https://github.com/fabian-sp/sda`. For our codebase, we use the official implementation of Rozet & Louppe (2023) as starting point.

B.1 Hyperparameters

An overview of the default hyperparameters is given in Table 1. Method-specific hyperparameter choices are listed thereafter.

Table 1: Default hyperparameter settings (if not specified otherwise).

Name	Default	Comment
Warmup	5 epochs	not used in Rozet & Louppe (2023)
Learning-rate schedule	linear-decay	`ScheduleFree` uses warmup+constant.
Gradient clipping	1.0	not used in Rozet & Louppe (2023)
Batch size	32	-
Epochs	1024	-
Momentum	0.9	applies to `SGD` and `Muon`
`AdamW` Betas	$(0.9, 0.999)$	applies to `AdamW`, `ScheduleFree`, `SOAP`

Table 2: Method-specific hyperparameters for `Muon`

Name	Value
Nesterov momentum	true
Newton-Schulz coefficients	$(3.4445, -4.7750, 2.0315)$
Newton-Schulz steps	5

Table 3: Method-specific hyperparameters for `SOAP`

Name	Value
Preconditioning frequency	10
Max preconditioning dimension	10^5

Momentum coefficients. For `SGD`, we use heavy-ball momentum with coefficient 0.9. We set `dampening` to 0.9 (this does not affect performance as it leads to an equivalent re-parametrization as long as the learning rate is tuned). For `AdamW`, `SOAP`, and `ScheduleFree`, we use always $(\beta_1, \beta_2) = (0.9, 0.999)$. For `Muon`, see below.

Details on `Muon` implementation. The core idea behind `Muon` is, for a weight matrix with gradient $G \in \mathbb{R}^{d_1 \times d_2}$, to compute (approximately) the closest orthogonal matrix G. It is given by UV^T, where $G = U\Sigma V^T$ is the singular value decomposition (Bernstein & Newhouse, 2025). This poses the question how to trainable parameters that are not 2-dimensional. Here, we follow the standard method proposed by Jordan et al. (2024): all bias and (time) embedding parameters are optimized with `AdamW`; for all parameters with more than two dimensions, we reshape their gradient into matrix shape, apply the Newton-Schulz algorithm, and reshape back to the original shape.[2] Moreover, in order to avoid separate tuning of the learning rate and weight decay for the `AdamW`-trained and the `Muon`-trained parameters, we apply the heuristic of Liu et al. (2025), which roughly aligns the update magnitude of the two methods, and therefore allows to use one single learning rate/weight decay.

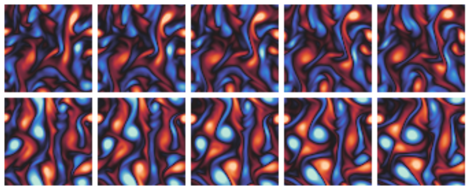
For `Muon`-trained parameters we use Nesterov momentum of 0.9; for `AdamW`-trained parameters we use $(\beta_1, \beta_2) = (0.9, 0.999)$.

Sampling hyperparameters. We set all hyperparameters that are not directly related to the training algorithm exactly as Rozet & Louppe (2023). In particular, they use a cosine schedule for the diffusion process. After training is completed, we sample two trajectories for 64 steps, always with the same seed.

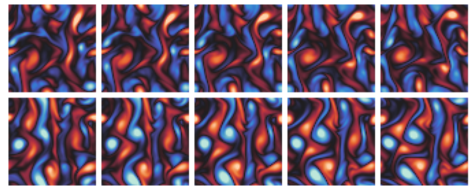
Details on schedule comparison. For the comparison of `wsd` and `cosine` schedules, we only tune the peak learning rate, and keep weight decay fixed at 10^{-3} (the original setting in Rozet & Louppe (2023)). For `wsd` we set the length of the cooldown to 20% of the total training, that is, cooldown starts after 819 epochs. For both `wsd` and `cosine` we cool down the learning rate to zero.

[2]This means that a parameters of shape $(d_0, \dots, d_m)$ will be reshaped into the shape $(d_0, \prod_{j=1}^{m} d_j)$.

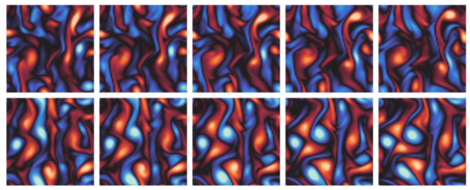
B.2 Additional Plots



(a) `AdamW`



(b) `Muon`



(c) `SOAP`

Figure 5: Vorticity of the generated velocity field, plotted for two trajectories with five snapshots each, after training completed. For each method, we display the hyperparameters that achieved lowest validation loss.

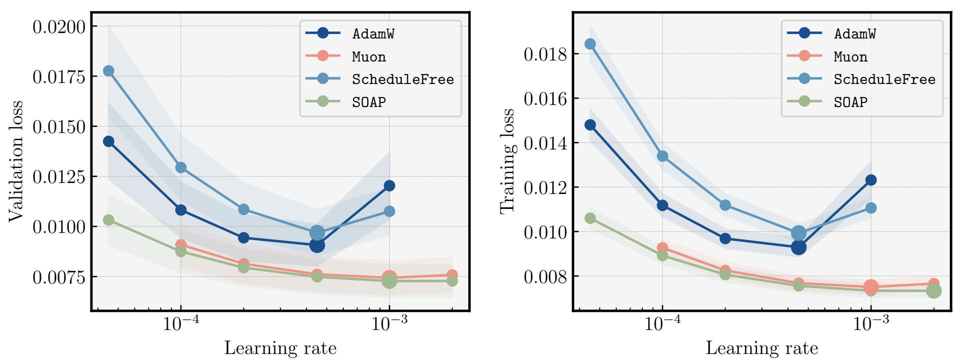


Figure 6: **(Left)** Same as Fig. 1, (left), but showing a band of one standard deviation over three runs. **(Right)** Same as (left), but for training loss.

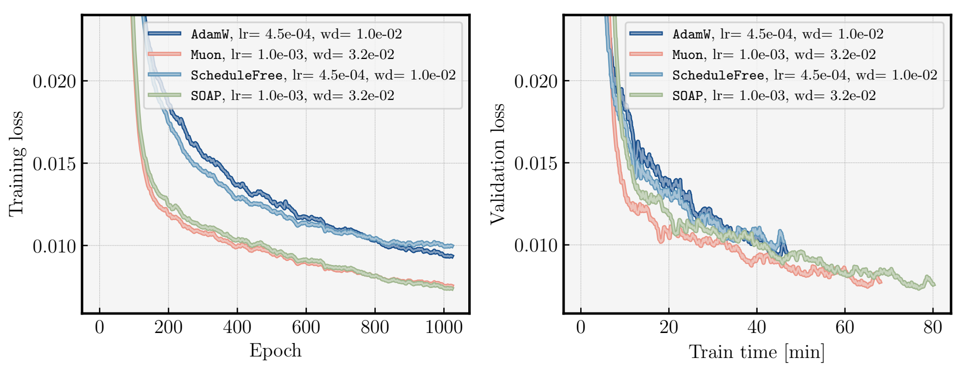


Figure 7: Training loss curve **(middle)** and validation loss curve with respect to train time for the best found setup for each method (minimal final validation loss). Legend indicates learning rate (lr) and weight decay (wd) values. To obtain smoother curves we plot exponential moving averages with coefficient 0.95.

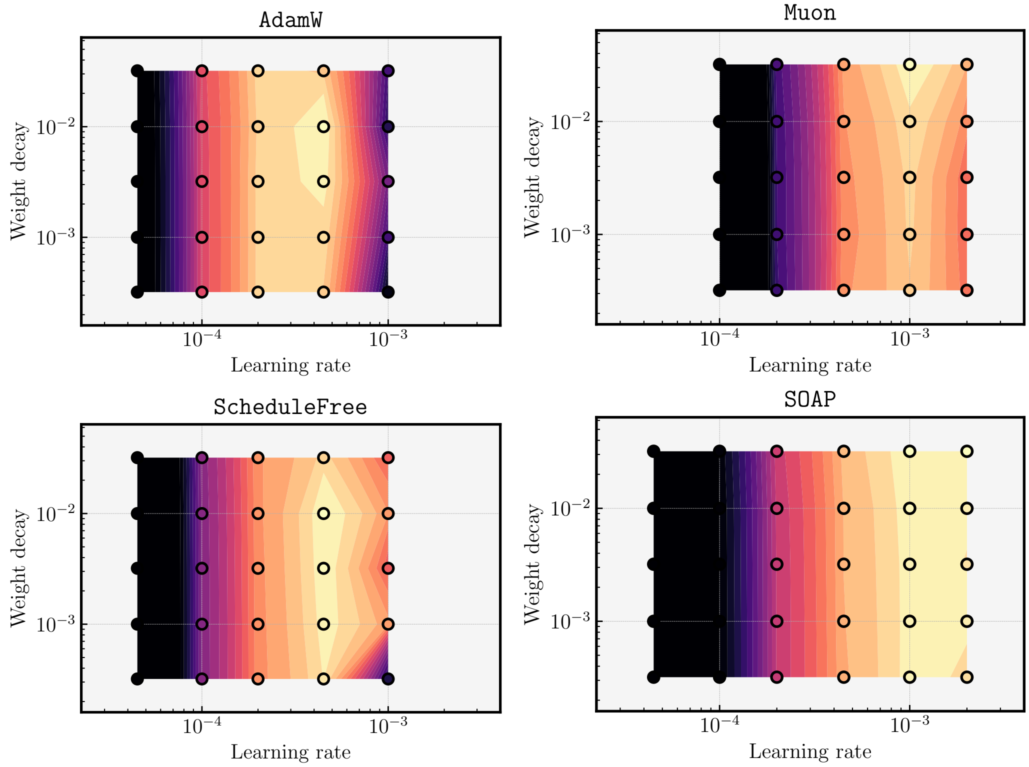
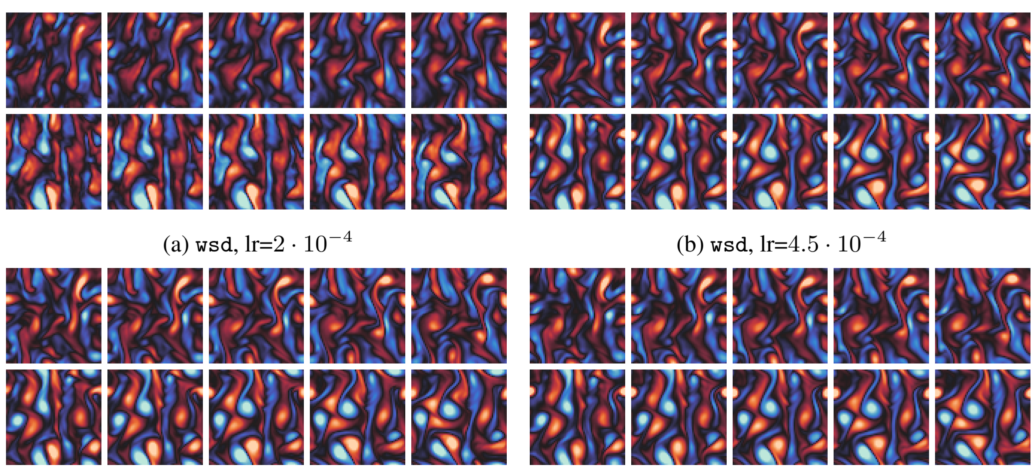


Figure 8: Heatmap of final validation loss (brighter is better) on the grid of learning rate and weight decay values. Each dot marks a hyperparameter combination that was run. Color indicates final validation loss (averaged over last five epochs), and color scale is different for each method in order to improve visibility.



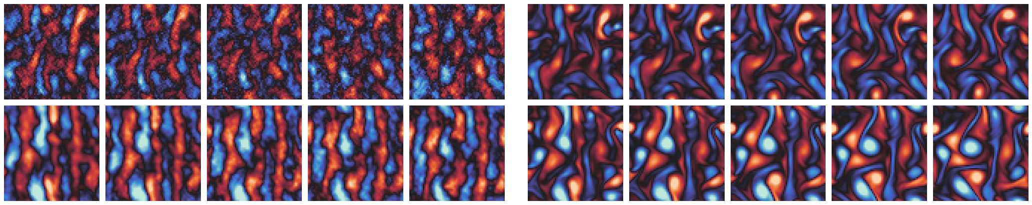
(a) `wsd`, lr=$2 \cdot 10^{-4}$

(b) `wsd`, lr=$4.5 \cdot 10^{-4}$

(c) `cosine`, lr=$2 \cdot 10^{-4}$

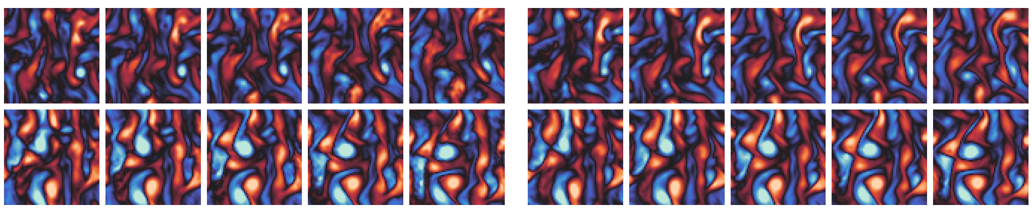
(d) `cosine`, lr=$4.5 \cdot 10^{-4}$

Figure 9: Vorticity of the generated velocity field, plotted for two trajectories with five snapshots each, after training completed. For `wsd`, the learning rate that achieves minimal validation loss (left) actually results in lower quality of the generated trajectories.



(a) ScheduleFree (train loss 0.00995) (b) AdamW (train loss 0.01022)

Figure 10: **For ScheduleFree, similar loss values do not result in similar generative quality.** Trajectories generated for the best ScheduleFree run, and a AdamW run with comparable, slightly higher, loss value. The quality of images generated with the model trained with ScheduleFree is significantly worse.



(a) ScheduleFree (train loss 0.01162) (b) ScheduleFree + wsd (train loss 0.01136)

Figure 11: **Learning-rate annealing on top of ScheduleFree improves generative quality.** For ScheduleFree, better loss values do not always correspond to better generative quality (compare **(left)** to Fig. 10 (left)). **(Right)** When adding the wsd schedule to ScheduleFree, the generative quality of the model improves (for some hyperparameter configurations). Here, we display learning rate=0.001 and weight decay=0.00032 (left and right).

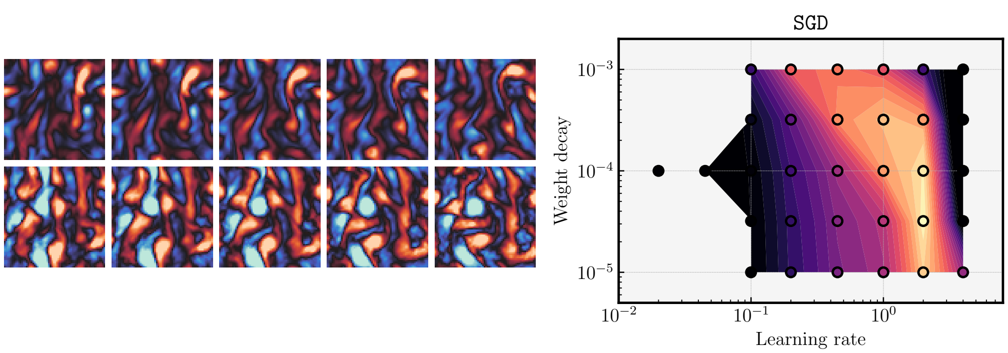


Figure 12: **(Left)** Vorticity of generated trajectories for the best setting we found for SGD. **(Right)** Heatmap of validation loss on the hyperparameter grid for SGD, for details see caption of Fig. 8.