

# DATA-DRIVEN MODELS OF PHYSICAL SYSTEMS: EFFICIENCY, CONSISTENCY, AND INFERENCE AT SCALE

Omer ROCHMAN SHARABI

*advised by*

Gilles LOUPPE

Submitted in partial fulfilment of the requirements  
for the degree of Doctor of Philosophy

*to the*

Department of Electrical Engineering and Computer Science  
University of Liège

July 31, 2026

*Solomon saith: There is no new thing upon the earth. So that as Plato had an imagination, that all knowledge was but remembrance; so Solomon giveth his sentence, that all novelty is but oblivion.*

Francis Bacon: *Essays*, LVIII.



# ABSTRACT

Data-driven models are emerging as a practical alternative to numerical simulation for forward and inverse modeling of physical systems, but they remain expensive to train and run, can violate the physics they represent, and rarely support inference at realistic scales. This thesis addresses all three limitations. NeuralMPM, a grid-based neural emulator for Lagrangian particle systems, cuts training time from days to hours and memory by up to two orders of magnitude while matching or exceeding the accuracy of graph-based emulators. Appa, a latent diffusion model of the global atmosphere trained on ERA5, compresses states by a factor of over 500 and treats reanalysis, filtering, and forecasting as posterior sampling from a single prior, conditioned on arbitrary observations of states with  $10^8$  components. For physical consistency, we show that training-free projections can enforce governing-equation constraints on neural PDE solvers while improving their accuracy, and we introduce Predict-Project-Renoise, a training-free algorithm for sampling pretrained diffusion models under hard constraints that respects the constrained distribution, and remains effective at the scale of Appa. Enforcing physics after the fact works when we know what to enforce; knowing what to enforce is the hard part, and physical structure must ultimately be learned from the data itself.

# JURY

Margaux BOXHO

Research Engineer at Cenaero

Patrick GALLINARI

Professor at Sorbonne University

Pierre GEURTS (President)

Professor at the University of Liège

Vân Anh HUYNH-THU

Assistant Professor at the University of Liège

Gilles LOUPPE (Advisor)

Professor at the University of Liège

Vlado MENKOVSKI

Associate Professor at Eindhoven University of Technology

# CONTENTS

|           |                                                                                                 |           |
|-----------|-------------------------------------------------------------------------------------------------|-----------|
| <b>I</b>  | <b>Prologue</b>                                                                                 | <b>1</b>  |
| <b>1</b>  | <b>Introduction</b>                                                                             | <b>2</b>  |
| 1.1       | Research questions . . . . .                                                                    | 5         |
| 1.2       | Outline and contributions . . . . .                                                             | 6         |
| 1.3       | Publications . . . . .                                                                          | 6         |
| <b>II</b> | <b>Papers</b>                                                                                   | <b>13</b> |
| <b>2</b>  | <b>A Neural Material Point Method for Particle-based Emulation</b>                              | <b>14</b> |
| 2.1       | Introduction . . . . .                                                                          | 15        |
| 2.2       | Computational Fluid Dynamics . . . . .                                                          | 15        |
| 2.3       | NeuralMPM . . . . .                                                                             | 17        |
| 2.4       | Experiments . . . . .                                                                           | 19        |
| 2.4.1     | Ablation study . . . . .                                                                        | 22        |
| 2.4.2     | Comparison with previous work . . . . .                                                         | 22        |
| 2.4.3     | Generalization . . . . .                                                                        | 24        |
| 2.4.4     | Inverse design problem . . . . .                                                                | 25        |
| 2.5       | Conclusion . . . . .                                                                            | 25        |
| 2.A       | Supplementary Material . . . . .                                                                | 30        |
| 2.B       | Training details . . . . .                                                                      | 30        |
| 2.C       | Additional Discussion . . . . .                                                                 | 31        |
| 2.D       | Supplementary results . . . . .                                                                 | 32        |
| 2.E       | Gallery of predicted trajectories . . . . .                                                     | 36        |
| <b>3</b>  | <b>Appa: Bending Weather Dynamics with Latent Diffusion Models for Global Data Assimilation</b> | <b>45</b> |
| 3.1       | Introduction . . . . .                                                                          | 46        |
| 3.2       | Appa . . . . .                                                                                  | 46        |
| 3.3       | Experiments . . . . .                                                                           | 47        |
| 3.4       | Discussion . . . . .                                                                            | 49        |
| 3.A       | Data . . . . .                                                                                  | 54        |
| 3.A.1     | ERA5 . . . . .                                                                                  | 54        |
| 3.A.2     | Data pre-processing . . . . .                                                                   | 54        |
| 3.B       | Technical details . . . . .                                                                     | 56        |
| 3.B.1     | Architectures . . . . .                                                                         | 56        |
| 3.B.2     | Generating trajectories . . . . .                                                               | 57        |
| 3.B.3     | Assimilation tasks . . . . .                                                                    | 57        |
| 3.C       | Evaluation metrics . . . . .                                                                    | 58        |
| 3.D       | Additional results . . . . .                                                                    | 59        |

|            |                                                                                                     |            |
|------------|-----------------------------------------------------------------------------------------------------|------------|
| 3.D.1      | Power spectral density . . . . .                                                                    | 59         |
| 3.D.2      | Physical consistency . . . . .                                                                      | 60         |
| 3.D.3      | Quantitative evaluation . . . . .                                                                   | 62         |
| 3.D.4      | Qualitative snapshots . . . . .                                                                     | 65         |
| <b>4</b>   | <b>Enforcing Governing Equation Constraints in Neural PDE Solvers via Training-Free Projections</b> | <b>71</b>  |
| 4.1        | Introduction . . . . .                                                                              | 71         |
| 4.2        | Projection operators . . . . .                                                                      | 72         |
| 4.3        | Experiments . . . . .                                                                               | 73         |
| 4.4        | Conclusion . . . . .                                                                                | 74         |
| 4.A        | Data . . . . .                                                                                      | 78         |
| 4.A.1      | LORENZ data . . . . .                                                                               | 78         |
| 4.A.2      | KURAMOTO-SIVASHINSKY data . . . . .                                                                 | 78         |
| 4.A.3      | NAVIER-STOKES data . . . . .                                                                        | 79         |
| 4.B        | Additional results . . . . .                                                                        | 79         |
| <b>5</b>   | <b>Predict-Project-Renoise: Sampling Diffusion Models under Hard Constraints</b>                    | <b>81</b>  |
| 5.1        | Introduction . . . . .                                                                              | 81         |
| 5.2        | Score-based generative modeling . . . . .                                                           | 82         |
| 5.3        | Method . . . . .                                                                                    | 83         |
| 5.3.1      | Constrained diffusion . . . . .                                                                     | 83         |
| 5.3.2      | Predict-Project-Renoise (PPR) sampling . . . . .                                                    | 84         |
| 5.4        | Related Work . . . . .                                                                              | 86         |
| 5.5        | Experiments . . . . .                                                                               | 87         |
| 5.6        | Conclusion . . . . .                                                                                | 91         |
| 5.A        | Metrics . . . . .                                                                                   | 94         |
| 5.B        | Data . . . . .                                                                                      | 94         |
| 5.B.1      | DATA2D . . . . .                                                                                    | 94         |
| 5.B.2      | KURAMOTO-SIVASHINSKY . . . . .                                                                      | 96         |
| 5.B.3      | Architecture . . . . .                                                                              | 96         |
| 5.B.4      | Training . . . . .                                                                                  | 97         |
| 5.B.5      | Appa . . . . .                                                                                      | 97         |
| 5.C        | Theoretical analysis and approximation error bound . . . . .                                        | 97         |
| 5.D        | Detailed results . . . . .                                                                          | 100        |
| 5.E        | Additional Ablations . . . . .                                                                      | 103        |
| 5.F        | Additional KS Constraints . . . . .                                                                 | 105        |
| 5.G        | Example samples from all experiments . . . . .                                                      | 110        |
| <b>III</b> | <b>Epilogue</b>                                                                                     | <b>119</b> |
| <b>6</b>   | <b>Conclusion</b>                                                                                   | <b>120</b> |
| 6.1        | Summary of contributions . . . . .                                                                  | 120        |
| 6.2        | On physical correctness . . . . .                                                                   | 121        |
| 6.3        | Future directions . . . . .                                                                         | 123        |
| 6.4        | Closing remarks . . . . .                                                                           | 123        |

# I

## PROLOGUE

# 1 INTRODUCTION

*Nous devons donc envisager l'état présent de l'univers, comme l'effet de son état antérieur, et comme la cause de celui qui va suivre.*

— Laplace, *Essai philosophique sur les probabilités*, 1814

Astronomy, one of the earliest sciences—and a driving force behind the development of Mathematics [1–3]—is descriptive and predictive in nature. It developed out of two very human impulses. The first, bewilderment for the natural world and the night sky. The second, and perhaps more important and practical, the life-or-death matter of the seasons, the harvests, the flooding of rivers, and the divine<sup>1</sup>. The common thread in the latter examples is that the prediction is used in a downstream task to plan or achieve a desired outcome in the real world, be it to increase the yield of the fields, to minimize the impact of cyclic catastrophes, or to decide whether to battle Hannibal<sup>2</sup>. Planning and control have always been inextricably linked to the predictive aspects of astronomy [4]. It then pays to realize that these aspects can be classified into two structurally distinct problem classes.

One is known as the *forward problem* and it asks “given a cause, what are its effects” [5, 6]. Many relevant problems fit in this definition, most notably time-dependent prediction—going from the current state of a system to its state at a future time—which is ubiquitous, e.g., the aforementioned astronomical and orbital motions, trajectories of cannonballs or of molecules, the development of waves in a pond after a stone has been thrown in it, or the propagation of pandemics. However, the forward problem in its general statement includes a larger class of problems, such as mapping parameters to observations (what we can measure): what is the acceleration at the surface of a planet given its mass, what shape will a suspended rope adopt given its tensile strength, or what will a thermometer read given the real temperature. While the forward problem is interesting in its own right, has a multitude of applications, and has deep epistemological implications, many problems do not fit this definition.

The other problem class is known as the *inverse problem* and its question is “given an effect, what was its cause” [5, 6]. Problems in this class include estimating the mass of a planet after measuring the acceleration on its surface, computing how infectious a disease is given its spread, estimating the initial speed and angle of a basketball given its final position, obtaining the real temperature of a room after looking at the measurement of a thermometer, or the reconstruction of MRI images from magnetic measurements. One remarkable property of many inverse problems is that they are *ill-posed*: a solution may not exist, may not be unique, or may depend unstably on small perturbations [7]. If you walk into a room and find a broken cup and coffee all over the floor, was it thrown by a cat, an earthquake, or perhaps a sour roommate? It is impossible to know *a priori* which one it is. When you have additional information, such as the fact that you do not have a cat and that earthquakes are very unlikely in Belgium, you could then guess a

<sup>1</sup> *What king or people has there ever been who did not employ divination?*

— Cicero, *De Divinatione* 1.95

<sup>2</sup> *The keeper of the sacred chickens advised the postponement of battle.*

— Cicero, *De Divinatione* 1.77

*In solving a problem of this sort, the grand thing is to be able to reason backwards.*

— Conan Doyle, *A Study in Scarlet*, 1887

*posteriori* that the probable answer is a roommate. If, however, you lived in Japan and got along with your roommate, the earthquake would be a plausible explanation.

In this thesis, we focus on physical systems, which are closely linked to one of the canonical inverse problems, that of *measurement*. Given an incomplete and noisy measurement (or observation) of a system, the task is to infer the full state of the system. Beyond this classic setting, the inverse problem umbrella also includes *inverse design* [8, 9]. Intuitively, one can observe the last state of a trajectory and ask what was the initial state. The answer will be the same as if we had asked what initial state we need to set up to obtain a desired end state. The best lift-to-drag ratio of an airfoil, optimal cooling profile of a chip, or target receptors for a protein can be thought of in the same way. The difference between inverse design and the traditional inverse problem framing is one of intentionality, not of mathematical structure.

Mapping causes to effects usually requires building a model. A model may be physical, built in the real world; conceptual, explaining the relation qualitatively; mathematical, formalized through expert knowledge and human labor; or computational, where a computer mimics the relation, often by numerically simulating a mathematical model. From here on, the term *model* refers to mathematical and computational models.

Traditional physics-based approaches to forward modeling [10, 11] are labor intensive and time consuming. They typically require domain expertise, an understanding of the underlying mechanism, observational and experimental data, and—sometimes—serendipity. After the model has been created, there are often limitations. Even when the mechanistic knowledge is available and a phenomenon is fully understood, evaluating the forward model at realistic scales or conditions can be prohibitively expensive or downright impossible<sup>3</sup> [12, 13]. Beyond the full-knowledge best-case scenario, some phenomena are not fully understood or we can measure a given state only partially and noisily [14, 15]. However, it is a tried-and-true method that has proved its worth and, until relatively recently [16–18], was the only way of solving the forward problem.

Recently, *learned* computational models, or machine learning (ML) models, and especially deep learning (DL), have emerged as viable alternatives [19–21]. Their rise has ushered in a new forward modeling paradigm, where instead of a human expert distilling mechanisms from data or experience<sup>4</sup>, a machine distills relations between cause and effect from data. For instance, the data could be a set of orbital observations for the planets, and a model can be trained to learn to predict their future positions based on current observations.

These learned models have already produced notable successes in forward modeling: weather prediction [22–27], partial differential equation (PDE) solving [20, 21, 28–32], molecular dynamics [33], protein folding [34], and fluid dynamics [35]. Learned surrogates are often cheaper to evaluate than the simulators they emulate, differentiable by construction, and able to extract patterns from data that a human modeler might miss or discard through approximation. Naturally [36, 37], they also have distinct limitations, notably poor generalization to unseen data [38, 39] and limited interpretability [40, 41].

Due to their ill-posedness, inverse problems are more nuanced. Several causes may explain the same effect, or none may do so exactly (e.g., due to noise). The Bayesian formulation naturally deals with this ambiguity [6, 42]. It treats the unknown cause  $x$  as a random variable and combines a prior  $p(x)$ , encoding knowledge or beliefs about the world, with a likelihood  $p(y | x)$ , encoding the forward process that links causes to effects  $y$ . The answer to the inverse problem is a full distribution, the posterior  $p(x | y)$ , over plausible causes rather than a single candidate. The prior is precisely the additional information invoked for the broken cup earlier in this chapter: that you own no cat and

... los Colegios de  
Cartógrafos levantaron un  
Mapa del Imperio, que  
tenía el tamaño del Imperio  
y coincidía puntualmente  
con él.

— Borges, *Del rigor en la  
ciencia*, 1960

<sup>3</sup> A drop of water contains roughly  $N \sim 10^{21}$  atoms, so storing each atom's position and velocity requires  $6N$  numbers. At 32 bits per number, this is  $6N \cdot 4 \sim 2.4 \times 10^{22}$  bytes. About  $2.4 \times 10^{10}$  terabytes.

<sup>4</sup> i.e., data.

that earthquakes are rare in Belgium<sup>5</sup>.

Once these philosophical foundations are established, posteriors have to be estimated and sampled. There are a number of ways to do so, which we classify according to the amount of structure they demand of the forward model. At the most permissive end, black-box methods require nothing but the ability to run the forward model: evolutionary and genetic algorithms [43, 44] search for good candidates through repeated evaluation, and approximate Bayesian computation compares simulated and observed data to approximate the posterior [45]. Their generality is paid for in efficiency, as they require many evaluations of a possibly expensive simulator. If, in addition, the unnormalized posterior density can be evaluated at any candidate solution, Markov chain Monte Carlo (MCMC) methods become available [46]. MCMC costs can be high, as every step of the chain costs at least one forward evaluation. Further, mixing deteriorates as the dimensionality grows. If the forward model is also differentiable, more methods become available: maximum a posteriori estimation by gradient descent, variational and adjoint-based methods such as those used in operational weather assimilation [47, 48], and gradient-informed samplers like Hamiltonian Monte Carlo [49]. On the downside, while gradients are usually informative, they can also be scarce. Simulators that are differentiable in principle may not be so in practice, and making them so can require extensive rewriting [50].

Many scientific simulators occupy an intermediate position between forward model and likelihood: they can be evaluated, but the likelihood they implicitly define cannot be, buried as it is under latent variables, stochastic branching, or just sheer complexity. This regime, common in modern science, is addressed by simulation-based inference (SBI) [51], of which approximate Bayesian computation, mentioned above, is the classical member. Modern approaches instead train models, typically neural networks, on simulated pairs of causes and observations to approximate the posterior, the likelihood, or their ratio [52–55]. Once trained, inference is amortized: new observations are processed at low cost. Current SBI methods have limitations of their own; they require large simulation budgets to train, they struggle when parameters grow beyond a few hundred dimensions,—though score-based approaches are steadily pushing this frontier [56–58]—and, like all learned models, their estimates can be unfaithful [59]. Finally, at the far end of the spectrum, when no simulator is available at all and only data is, a surrogate forward model, or a direct inverse map, can be learned from existing pairs of causes and effects [60, 61].

Every method on this spectrum relies, at least implicitly, on some knowledge of the forward causal process (a well-understood analytical model and pure data are the two extremes), so the difficulties of the forward problem compound those native to the inverse problem, and errors in the forward model, arising from incomplete knowledge of the phenomenon, partial observations, or noisy measurements, propagate into the inverse solution. Remarkably, learned models can move problems along the spectrum. A neural emulator turns an expensive simulator into a cheap one, and a non-differentiable one into a differentiable one, unlocking methods that were previously out of reach.

The distributions required by the Bayesian framework are often over a very high-dimensional space, e.g., a global atmospheric state, a fluid velocity field, the positions of millions of particles. Representing such distributions and sampling from them is a problem where deep generative models have come into their own [62, 63]. Among model families, score-based or diffusion models [64, 65] are the current state of the art: they learn the score of the data distribution and generate samples by simulating a stochastic differential equation (SDE). Two of their properties make them exceptionally well suited to inverse problems. First, a diffusion model trained on states of a physical system is a prior in the Bayesian sense, a model of what plausible states look like. Second, sampling can be guided at inference time, without retraining, turning an

<sup>5</sup> Move to Japan, changing the prior, and the same likelihood yields a different posterior.

*... the curse of dimensionality, a malediction that has plagued the scientist from the earliest days.*

— Bellman, *Adaptive Control Processes*, 1961



unconditional prior model into a posterior one for, in principle, arbitrary observation processes [57, 66]. A single learned model thus serves both faces of the problem: sampled unconditionally, or conditionally on parameters or initial states, it is a forward model; guided by observations or design objectives, it solves the inverse problem. This recipe has led to notable successes in data assimilation [57], including oceanic [67], global atmospheric [68, 69], and kilometer-scale weather systems [70], in medical imaging [71] and astrophysics [58], and in inverse design, from proteins [72] and materials [73] to systems governed by PDEs [74].

Despite their notable success, data-driven models have their own issues. While learned surrogates are cheaper than simulators, they can still be too slow and memory-hungry to train and run in high-dimensional settings<sup>6</sup>. Additionally, they often fail to maintain physical consistency: they may fit observed data while violating properties the underlying systems are known to possess, such as conservation laws, constraints, symmetries, or governing dynamics [12, 75, 76]. Capturing multiscale phenomena is another persistent challenge [77–79], since many methods accurately model large-scale structures but fail to resolve small-scale behavior. Scaling inference itself to high-dimensional states also remains out of reach for MCMC and current SBI methods. These limitations matter increasingly as data-driven methods become competitive beyond research settings and begin to be applied in the real world. Addressing them is therefore essential and could improve the reliability of both forward and inverse modeling. This thesis focuses on data-driven models of physical systems, where these problems are especially important.

<sup>6</sup> A global ERA5 atmospheric state at 0.25° resolution has on the order of  $10^8$  components.

*... big whirls have little  
whirls that feed on their  
velocity, and little whirls  
have lesser whirls and so  
on...*

— Richardson, *Weather  
Prediction by Numerical  
Process*, 1922

## 1.1 RESEARCH QUESTIONS

Spurred by the limitations of current data-driven methods in physical systems and the potential that overcoming them offers, we explored the efficiency bottleneck in forward problems, the consistency issue, and how to improve sampling in Bayesian inverse problems.

**RQ1 (Efficiency).** How can learned forward models of complex physical systems be made more efficient to train and evaluate? Learned surrogates are introduced to escape the cost of numerical simulation, but state-of-the-art emulators are themselves expensive to train and to run, and the cost grows with the size of the state. Chapters 2 and 3 address this question at two very different scales, the former through a grid-based reformulation of particle emulation, the latter through generation in a learned latent space over  $500\times$  smaller than the state space.

**RQ2 (Consistency and sampling).** How can known physical constraints be enforced on learned models, without retraining them and without degrading their accuracy or sampled distribution? Post hoc enforcement decouples constraint satisfaction from training, but naive projections distort predictions and, in the generative case, bias the distribution. Chapters 4 and 5 address this question, the former for deterministic neural PDE solvers, the latter for score-based generative models, where correctness means respecting the constrained distribution itself.

**RQ3 (Scale).** Can Bayesian inverse problems be solved at the scale of real physical systems, with flexible conditioning on arbitrary observations? Classical assimilation methods rely on linearization and adjoints that are costly at a global scale, while current SBI methods do not reach these dimensions. Chapters 3 and 5 address this question in various settings, including global atmospheric modeling with states of dimension  $\sim 10^8$ .

## 1.2 OUTLINE AND CONTRIBUTIONS

The remainder of this thesis consists of four publications, followed by a conclusion. Chapter 2 presents NeuralMPM, a material point method (MPM)-inspired neural emulation framework for Lagrangian particle systems that interpolates particles onto a fixed grid and computes dynamics with grid-to-grid networks, reducing training time from days to hours and memory consumption by one to two orders of magnitude compared to graph network-based emulators, while matching or exceeding their long-term accuracy. Chapter 3 introduces Appa, a planetary-scale latent diffusion model of the global atmosphere trained on ERA5. Appa is a score-based generative model of atmospheric trajectories and as such it can be conditioned on arbitrary observations at inference time, so that reanalysis, filtering, and forecasting become special cases of posterior sampling. Appa also serves as a test-bed for Chapter 5. Chapter 4 studies training-free, post hoc projection strategies to enforce governing-equation constraints (i.e., PDE-based constraints) on neural solvers, finding that nonlinear projections anchored at the neural solver output can reduce constraint violation by orders of magnitude while improving accuracy over physics-informed baselines. Chapter 5 introduces Predict-Project-Renoise (PPR), a training-free method for sampling from pretrained diffusion models under hard constraints, which respects the constrained distribution while obtaining constraint violations orders of magnitude lower than similar methods on a diverse range of experiments, including Appa. Finally, Chapter 6 gives remarks and closing thoughts on the field as a whole and future directions.

## 1.3 PUBLICATIONS

The following publications form the core of this thesis, on all of which I played a lead role:

Omer Rochman-Sharabi, Sacha Lewin, and Gilles Louppe. “A Neural Material Point Method for Particle-based Emulation”. In *Transactions on Machine Learning Research* (2025)

Omer Rochman and Gilles Louppe. “Enforcing governing equation constraints in neural PDE solvers via training-free projections”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025

Omer Rochman-Sharabi and Gilles Louppe. “Predict-Project-Renoise: Sampling Diffusion Models under Hard Constraints”. 2026. Submitted

This thesis also draws on the following publication, which was a collaborative effort by the SAIL Montefiore Unit, led by G r me Andry:

G r me Andry, Fran ois Rozet, Sacha Lewin, Omer Rochman, Victor Mangeleer, Matthias Pirlet, Elise Faulx, Marilaure Gr goire, and Gilles Louppe. “Appa: Bending Weather Dynamics with Latent Diffusion Models for Global Data Assimilation”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025

During my PhD, I also led and contributed to a number of other works:

Omer Rochman-Sharabi and Gilles Louppe. “Differentiable composition for model discovery”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2022

Omer Rochman-Sharabi and Gilles Louppe. “Trick or treat? Evaluating stability strategies in graph network-based simulators”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2023

### 1.3 PUBLICATIONS

Carles Cantero, Isidro Gomez, Omer Rochman, Damien Ségransan, and Xavier Dumusque. “SpeckleNet: a large-scale PSF subtraction deep learning model for exoplanet detection and characterization in high contrast imaging”. In *EPSC-DPS Joint Meeting 2025*. 2025, EPSC-DPS2025–2087

## REFERENCES

- [1] Morris Kline. “Mathematical Thought from Ancient to Modern Times”. New York: Oxford University Press, 1972.
- [2] Carl B. Boyer and Uta C. Merzbach. “A History of Mathematics”. 3rd ed. Hoboken, NJ: John Wiley & Sons, 2011.
- [3] Encyclopaedia Britannica. “Astronomy: History of Astronomy”. <https://www.britannica.com/science/astronomy/History-of-astronomy>. Accessed 21 May 2026.
- [4] European Space Agency. “A History of Astrometry, Part I: Mapping the Sky from Ancient to Pre-modern Times”. <https://sci.esa.int/web/gaia/-/53196-the-oldest-sky-maps>. Accessed 21 May 2026.
- [5] Marko Vauhkonen, Tanja Tarvainen, and Timo Lähivaara. “Inverse Problems”. In *Mathematical Modelling*. Ed. by Seppo Pohjolainen. Cham: Springer, 2016, pp. 207–227.
- [6] Albert Tarantola. “Inverse Problem Theory and Methods for Model Parameter Estimation”. Philadelphia: Society for Industrial and Applied Mathematics, 2005.
- [7] Jacques Hadamard. “Sur les problèmes aux dérivées partielles et leur signification physique”. In *Princeton University Bulletin* 13 (1902), pp. 49–52.
- [8] Martin Burger, Stanley J. Osher, and Eli Yablonovitch. “Inverse Problem Techniques for the Design of Photonic Crystals”. In *IEICE Transactions on Electronics* E87-C.3 (2004), pp. 258–265.
- [9] Luka Grbcic, Juliane Müller, and Wibe Albert de Jong. “Efficient Inverse Design Optimization Through Multi-Fidelity Simulations, Machine Learning, and Boundary Refinement Strategies”. In *Engineering with Computers* 40 (2024), pp. 4081–4108.
- [10] Anuj Karpatne, Xiaowei Jia, and Vipin Kumar. “Knowledge-guided Machine Learning: Current Trends and Future Prospects”. In *arXiv preprint arXiv:2403.15989* (2024).
- [11] Jared Willard et al. “Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems”. In *ACM Computing Surveys* 55.4 (2022), pp. 1–37.
- [12] George Em Karniadakis et al. “Physics-informed Machine Learning”. In *Nature Reviews Physics* 3 (2021), pp. 422–440.
- [13] Rainer Friedrich et al. “Direct Numerical Simulation of Incompressible Turbulent Flows”. In *Computers & Fluids* 30.5 (2001), pp. 555–579.
- [14] Patrick A. K. Reinbold et al. “Robust Learning from Noisy, Incomplete, High-Dimensional Experimental Data via Physically Constrained Symbolic Regression”. In *Nature Communications* 12.1 (2021), p. 3219.
- [15] Ward Quaghebeur, Ingmar Nopens, and Bernard De Baets. “Incorporating Unmodeled Dynamics into First-Principles Models Through Machine Learning”. In *IEEE Access* 9 (2021), pp. 22014–22022.
- [16] I. E. Lagaris, A. Likas, and D. I. Fotiadis. “Artificial Neural Networks for Solving Ordinary and Partial Differential Equations”. In *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000.
- [17] G. Berkooz, P. Holmes, and J. L. Lumley. “The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows”. In *Annual Review of Fluid Mechanics* 25 (1993).

- [18] Peter Benner, Serkan Gugercin, and Karen Willcox. “A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems”. In *SIAM Review* 57.4 (2015).
- [19] Giuseppe Carleo et al. “Machine Learning and the Physical Sciences”. In *Reviews of Modern Physics* 91.4 (2019), p. 045002.
- [20] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations”. In *Journal of Computational Physics* 378 (2019), pp. 686–707.
- [21] Zongyi Li et al. “Fourier Neural Operator for Parametric Partial Differential Equations”. In *International Conference on Learning Representations*. 2021.
- [22] Jaideep Pathak et al. “FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators”. In *arXiv preprint arXiv:2202.11214* (2022).
- [23] Kaifeng Bi et al. “Accurate Medium-Range Global Weather Forecasting with 3D Neural Networks”. In *Nature* 619 (2023), pp. 533–538.
- [24] Remi Lam et al. “Learning Skillful Medium-Range Global Weather Forecasting”. In *Science* 382.6677 (2023), pp. 1416–1421.
- [25] Simon Lang et al. “AIFS – ECMWF’s Data-Driven Forecasting System”. 2024.
- [26] Ilan Price et al. “Probabilistic Weather Forecasting with Machine Learning”. In *Nature* 637 (2025), pp. 84–90.
- [27] Cristian Bodnar et al. “A Foundation Model for the Earth System”. In *Nature* 641 (2025), pp. 1180–1187.
- [28] Lu Lu et al. “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators”. In *Nature Machine Intelligence* 3.3 (2021).
- [29] Nikola Kovachki et al. “Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs”. In *Journal of Machine Learning Research* 24.89 (2023).
- [30] Zongyi Li et al. “Physics-Informed Neural Operator for Learning Partial Differential Equations”. In *ACM / IMS Journal of Data Science* 1.3 (2024), pp. 1–27.
- [31] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. “Message Passing Neural PDE Solvers”. In *International Conference on Learning Representations*. 2022.
- [32] Phillip Lippe et al. “PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers”. In *Advances in Neural Information Processing Systems*. Vol. 36. 2023.
- [33] Frank Noé et al. “Machine Learning for Molecular Simulation”. In *Annual Review of Physical Chemistry* 71 (2020), pp. 361–390.
- [34] John Jumper et al. “Highly Accurate Protein Structure Prediction with AlphaFold”. In *Nature* 596.7873 (2021), pp. 583–589.
- [35] Dmitrii Kochkov et al. “Machine Learning-Accelerated Computational Fluid Dynamics”. In *Proceedings of the National Academy of Sciences* 118.21 (2021), e2101784118.
- [36] David H. Wolpert. “The Lack of A Priori Distinctions Between Learning Algorithms”. In *Neural Computation* 8.7 (1996), pp. 1341–1390.
- [37] David H. Wolpert and William G. Macready. “No Free Lunch Theorems for Optimization”. In *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82.

- [38] Pang Wei Koh et al. “WILDS: A Benchmark of in-the-Wild Distribution Shifts”. In *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 5637–5664.
- [39] Han Yu et al. “A Survey on Evaluation of Out-of-Distribution Generalization”. In *arXiv preprint arXiv:2403.01874* (2024).
- [40] Cynthia Rudin. “Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead”. In *Nature Machine Intelligence* 1 (2019), pp. 206–215.
- [41] Finale Doshi-Velez and Been Kim. “Towards A Rigorous Science of Interpretable Machine Learning”. In *arXiv preprint arXiv:1702.08608* (2017).
- [42] Andrew M. Stuart. “Inverse Problems: A Bayesian Perspective”. In *Acta Numerica* 19 (2010), pp. 451–559.
- [43] Nikolaus Hansen. “The CMA Evolution Strategy: A Tutorial”. In *arXiv preprint arXiv:1604.00772* (2016).
- [44] Darrell Whitley. “A Genetic Algorithm Tutorial”. In *Statistics and Computing* 4 (1994), pp. 65–85.
- [45] Scott A. Sisson, Yanan Fan, and Mark A. Beaumont. “Handbook of Approximate Bayesian Computation”. Chapman and Hall/CRC, 2018.
- [46] Klaus Mosegaard and Albert Tarantola. “Monte Carlo Sampling of Solutions to Inverse Problems”. In *Journal of Geophysical Research: Solid Earth* 100.B7 (1995), pp. 12431–12447.
- [47] Dan Givoli. “A Tutorial on the Adjoint Method for Inverse Problems”. In *Computer Methods in Applied Mechanics and Engineering* 380 (2021), p. 113810.
- [48] P. Courtier, J.-N. Thépaut, and A. Hollingsworth. “A Strategy for Operational Implementation of 4D-Var, Using an Incremental Approach”. In *Quarterly Journal of the Royal Meteorological Society* 120.519 (1994), pp. 1367–1387.
- [49] Michael Betancourt. “A Conceptual Introduction to Hamiltonian Monte Carlo”. 2018.
- [50] William S. Moses and Valentin Churavy. “Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients”. In *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 12472–12485.
- [51] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. “The frontier of simulation-based inference”. In *Proceedings of the National Academy of Sciences* 117.48 (2020).
- [52] George Papamakarios, David C. Sterratt, and Iain Murray. “Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows”. 2019.
- [53] Joeri Hermans, Volodimir Begy, and Gilles Louppe. “Likelihood-free MCMC with Amortized Approximate Ratio Estimators”. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020.
- [54] Manuel Gloeckler et al. “All-in-one simulation-based inference”. In *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. PMLR, 2024, pp. 15735–15766.
- [55] Louis Sharrock et al. “Sequential Neural Score Estimation: Likelihood-Free Inference with Conditional Score Based Diffusion Models”. In *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. PMLR, 2024, pp. 44565–44602.

- [56] Ronan Legin et al. “Posterior sampling of the initial conditions of the universe from non-linear large scale structures using score-based generative models”. In *Monthly Notices of the Royal Astronomical Society: Letters* 527.1 (2023), pp. L173–L178.
- [57] François Rozet and Gilles Louppe. “Score-based Data Assimilation”. In *Advances in Neural Information Processing Systems*. Vol. 36. 2023.
- [58] Alexandre Adam et al. “Posterior samples of source galaxies in strong gravitational lenses with score-based priors”. 2022.
- [59] Joeri Hermans et al. “A Trust Crisis In Simulation-Based Inference? Your Posterior Approximations Can Be Unfaithful”. In *Transactions on Machine Learning Research* (2022).
- [60] Shima Kamyab et al. “Deep learning methods for inverse problems”. In *PeerJ Computer Science* 8 (2022), e951.
- [61] Simon Arridge et al. “Solving Inverse Problems Using Data-Driven Models”. In *Acta Numerica* 28 (2019), pp. 1–174.
- [62] Jakub M. Tomczak. “Deep Generative Modeling: From Probabilistic Framework to Generative AI”. In *Entropy* 27.3 (2025).
- [63] Jakub M Tomczak. “Deep Generative Modeling”. Springer Cham, 2024.
- [64] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising Diffusion Probabilistic Models”. In *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020.
- [65] Yang Song et al. “Score-Based Generative Modeling through Stochastic Differential Equations”. In *International Conference on Learning Representations*. 2021.
- [66] Giannis Daras et al. “A Survey on Diffusion Models for Inverse Problems”. 2024.
- [67] Scott A. Martin, Georgy E. Manucharyan, and Patrice Klein. “Generative Data Assimilation for Surface Ocean State Estimation From Multi-Modal Satellite Observations”. In *Journal of Advances in Modeling Earth Systems* 17.8 (2025), e2025MS005063.
- [68] G r me Andry et al. “Appa: Bending Weather Dynamics with Latent Diffusion Models for Global Data Assimilation”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025.
- [69] Langwen Huang et al. “DiffDA: a Diffusion model for weather-scale Data Assimilation”. In *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024.
- [70] Peter Manshausen et al. “Generative Data Assimilation of Sparse Weather Station Observations at Kilometer Scales”. 2025.
- [71] Yang Song et al. “Solving Inverse Problems in Medical Imaging with Score-Based Generative Models”. In *International Conference on Learning Representations*. 2022.
- [72] Joseph L. Watson et al. “De novo design of protein structure and function with RFdiffusion”. In *Nature* 620 (2023), pp. 1089–1100.
- [73] Claudio Zeni et al. “MatterGen: a generative model for inorganic materials design”. 2023.
- [74] Tailin Wu et al. “Compositional Generative Inverse Design”. 2024.
- [75] Chuizheng Meng et al. “When Physics Meets Machine Learning: A Survey of Physics-Informed Machine Learning”. In *Machine Learning for Computational Science and Engineering* 1 (2025), p. 20.

- [76] Omer Rochman and Gilles Louppe. “Enforcing governing equation constraints in neural PDE solvers via training-free projections”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025.
- [77] Phong C. H. Nguyen et al. “Challenges and Opportunities for Machine Learning in Multiscale Computational Modeling”. In *Journal of Computing and Information Science in Engineering* 23.6 (2023), p. 060808.
- [78] Benjamin Sanderse et al. “Scientific Machine Learning for Closure Models in Multiscale Problems: A Review”. In *Foundations of Data Science* 7.1 (2025), pp. 298–337.
- [79] Xinliang Liu et al. “Mitigating Spectral Bias for the Multiscale Operator Learning”. In *Journal of Computational Physics* 506 (2024), p. 112944.
- [80] Omer Rochman-Sharabi, Sacha Lewin, and Gilles Louppe. “A Neural Material Point Method for Particle-based Emulation”. In *Transactions on Machine Learning Research* (2025).
- [81] Omer Rochman-Sharabi and Gilles Louppe. “Predict-Project-Renoise: Sampling Diffusion Models under Hard Constraints”. 2026. Submitted.
- [82] Omer Rochman-Sharabi and Gilles Louppe. “Differentiable composition for model discovery”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2022.
- [83] Omer Rochman-Sharabi and Gilles Louppe. “Trick or treat? Evaluating stability strategies in graph network-based simulators”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2023.
- [84] Carles Cantero et al. “SpeckleNet: a large-scale PSF subtraction deep learning model for exoplanet detection and characterization in high contrast imaging”. In *EPSC-DPS Joint Meeting 2025*. 2025, EPSC-DPS2025–2087.



# II

## PAPERS

## 2 A NEURAL MATERIAL POINT METHOD FOR PARTICLE-BASED EMULATION

*And the craft createth a semblance.*

— Morris, *Sigurd the Volsung*

### ADDENDUM

This chapter is based on the paper:

Omer Rochman-Sharabi, Sacha Lewin, and Gilles Louppe. “A Neural Material Point Method for Particle-based Emulation”. In *Transactions on Machine Learning Research* (2025)

As lead author, Omer originated the method. Omer and Sacha shared its implementation, the experimental work, and the writing of the manuscript. Gilles supervised the project and took part in the writing.

For the version presented in this chapter, we have edited the structure and formatting of the manuscript to be consistent with the rest of this document. The content is otherwise that of the published paper.

### ABSTRACT

Mesh-free Lagrangian methods are widely used for simulating fluids, solids, and their complex interactions due to their ability to handle large deformations and topological changes. These physics simulators, however, require substantial computational resources for accurate simulations. To address these issues, deep learning emulators promise faster and scalable simulations, yet they often remain expensive and difficult to train, limiting their practical use. Inspired by the Material Point Method (MPM), we present NeuralMPM, a neural emulation framework for particle-based simulations. NeuralMPM interpolates Lagrangian particles onto a fixed-size grid, computes updates on grid nodes using image-to-image neural networks, and interpolates back to the particles. Similarly to MPM, NeuralMPM benefits from the regular voxelized representation to simplify the computation of the state dynamics, while avoiding the drawbacks of mesh-based Eulerian methods. We demonstrate the advantages of NeuralMPM on 6 datasets, including fluid dynamics and fluid-solid interactions simulated with MPM and Smoothed Particle Hydrodynamics (SPH). Compared to GNS and DMCF, NeuralMPM reduces training time from 10 days to 15 hours, memory consumption by 10x-100x, and increases inference speed by 5x-10x, while achieving comparable or superior long-term accuracy, making it a promising approach for practical forward and inverse problems. A project page is

available at <https://neuralmpm.isach.be>.

## 2.1 INTRODUCTION

The Navier-Stokes equations describe the time evolution of fluids and their interactions with solid materials. As analytical solutions rarely exist, numerical methods are required to approximate the solutions. On the one hand, Eulerian methods discretize the fluid domain on a fixed grid, simplifying the computation of the dynamics, but requiring high-resolution meshes to solve small-scale details in the flow. Lagrangian methods, on the other hand, represent the fluid as virtual moving particles defining the system’s state, hence maintaining a high level of detail in regions of high density. While effective at handling deformations and topological changes [2], Lagrangian methods struggle with collisions and interactions with rigid objects [3, 4].

Regardless of the discretization strategy, large-scale high-resolution numerical simulations are computationally expensive, limiting their practical use in downstream tasks such as forecasting, inverse problems, or computational design. To address these issues, deep learning emulators have shown promise in accelerating simulations by learning an emulator model that can predict the system’s state at a fraction of the cost. Next to their speed, neural emulators also have the strategic advantage of being differentiable, enabling their use in inverse problems and optimization tasks [5–7]. Moreover, they have the potential to be learned directly from real data, bypassing the costly and resource-intensive process of building a simulator [8–12]. In this direction, particle-based neural emulators [13–15] have seen success in accurately simulating fluids and generalizing to unseen environments. These emulators, however, suffer from the same issues as traditional Lagrangian methods, with collisions and interactions with rigid objects being particularly challenging. These emulators can also require long training and inference times, limiting their practical use.

Taking inspiration from the hybrid Material Point Method (MPM) [16, 17] that combines the strengths of both Eulerian and Lagrangian methods, we introduce NeuralMPM, a neural emulation framework for particle-based simulations.

- As in MPM, NeuralMPM uses Lagrangian particles to represent the system’s state but models system dynamics on voxelized grids. This approach benefits from a regular grid structure for simplified state dynamics computation while avoiding the drawbacks of mesh-based Eulerian methods.
- It interpolates particles onto a fixed-size grid, bypassing the need for expensive neighbor searches at each timestep and replacing them with two efficient voxelization-based interpolation steps [18].
- By defining dynamics on a grid, NeuralMPM leverages well-established grid-to-grid neural architectures. This introduces an inductive bias that enables easier processing of global and local point cloud structures, freeing capacity to learn the system’s dynamics.

Compared to previous data-driven approaches [13–15], these improvements reduce the training time from days to hours, while achieving higher or comparable accuracy.

## 2.2 COMPUTATIONAL FLUID DYNAMICS

Computational fluid dynamics simulations can be classified into two broad categories, Eulerian and Lagrangian, depending on the discretization of the fluid [19]. In Eulerian simulations, the domain is discretized with a mesh, with state variables  $u_i^t$  (such as mass

or momentum) maintained at each mesh point  $i$ . Well-known examples of Eulerian simulations are the finite difference method, where the domain is divided into a uniform regular grid (also called an Eulerian grid), and the finite element method, where the domain is divided into regions, or elements, that may have different shapes and density, allowing to increase the resolution in only some areas of the domain [20, 21]. Another widely used family of Eulerian methods is the spectral and pseudo-spectral family of methods, where the equation is solved in Fourier space or alternating between Fourier and real space, respectively, leveraging the fact that differentiation in real space is multiplication in Fourier space [22]. Lagrangian simulations, on the other hand, discretize the fluid as a set of virtual moving particles  $\{p_i^t, u_i^t\}_{i=1}^N$ , each described by its position  $p_i^t$  and state variables  $u_i^t$  that include the particle velocity  $v_i^t$ . To simulate the fluid, the particles move according to the dynamics of the system, producing a new set of particles  $\{p_i^{t+1}, u_i^{t+1}\}_{i=1}^N$  at each timestep. Simulations in Lagrangian coordinates are particularly useful when the fluid is highly deformable, as the particles can move freely and adapt to the fluid's shape. Among Lagrangian methods, Smoothed Particle Hydrodynamics (SPH) [23, 24] is one of the most popular, where the fluid is represented by a set of particles that interact with each other through a kernel function that smooths the interactions.

Hybrid Eulerian-Lagrangian methods combine the strengths of both frameworks. Like Lagrangian methods, they carry the system state information via particles, thereby automatically adjusting the resolution to the local density of the system. By using a regular grid, however, they simplify gradient computation, make entity contact detection easier, and prevent cracks from propagating only along the mesh. Among hybrid methods, the Material Point Method has gained popularity for its ability to handle large deformations and topological changes. MPM combines a regular Eulerian grid with moving Lagrangian particles. It does so in four main steps: (1) the quantities carried by the particles are interpolated onto a regular grid  $G^t = \mathbf{p2g}(\{p_i^t, u_i^t\})$  using a particle-to-grid (**p2g**) function, (2) the equations of motion are solved on the grid, where derivatives and other quantities are easier to compute, resulting in a new grid state  $G^{t+1} = f(G^t)$ , (3) the resulting dynamics are interpolated back onto the particles as  $\{u_i^{t+1}\} = \mathbf{g2p}(G^{t+1}, \{p_i^t\})$ , using a grid-to-particle (**g2p**) function, (4) the positions of the particles are updated by computing particle-wise velocities and using an appropriate integrator, such as Euler, i.e.,  $p_i^{t+1} = p_i^t + \Delta t v_i^{t+1}$ . The grid values are then reset for the next step. MPM has been used in soft tissue simulations [25], in molecular dynamics [26], in astrophysics [27], in fluid-membrane interactions [28], and in simulating cracks [29] and landslides [30]. MPM is also widely used in the animation industry, perhaps most notably in Disney's 2013 film Frozen [31], where it was used to simulate snow.

Notwithstanding the success of numerical simulators, they remain expensive, slow, and, most of the time, non-differentiable. In recent years, differentiable neural emulators have shown great promise in accelerating fluid simulations, most notably in a series of works to emulate SPH simulations in a fully data-driven manner. Graph network-based simulators (GNS) [13] use a graph neural network (GNN) and a graph built from the local neighborhood of the particles to predict the acceleration of the system. The approach requires building a graph out of the point cloud at every timestep to obtain structural information about the cloud, which is an expensive operation. In addition, the GNN needs to extract global information from its nodes, which is only possible with a high number of message-passing steps, resulting in a large computational graph and long training and inference times. This large computational graph, along with repeated construction, makes fully autoregressive training over long rollouts impractical, as the gradients need to backpropagate all the way back to the first step. Cheaper strategies exist, like the push-forward trick [32], but they have been shown to be inferior to fully backpropagating through trajectories [33, 34]. As autoregressive training is not available, the stability of the learned dynamics can be compromised, making the model prone to diverging

or oscillating. Noise injection training strategies can be used to increase the stability of the rollouts, but the magnitude of the noise becomes a critical parameter. Related approaches to GNS include Han et al. [35], who introduce improvements to GNS to make them subequivariant to certain transformations and show increased accuracy on simulations involving solid objects, and Viswanath et al. [36], who apply GNS on a reduced representation of the particle system obtained by farthest point sampling.

An alternative approach is the continuous convolution (CConv) [14, 37], an extension of convolutional networks to point clouds. In this method, a convolutional kernel is applied to each particle by interpolating the values of the kernel at the positions of its neighbors, which are found via spatial hashing on GPU, a cheaper alternative to tree-based searches that allows for autoregressive training. In [15], Deep Momentum Conserving Fluids (DMCF) build upon CConv to design a momentum-conserving architecture. Nevertheless, to account for long-range interactions, the authors introduce different branches, with different receptive fields, into their network. The number of branches, and their hyperparameters, need to be tuned to capture global dependencies, leading to long training times even with optimized CUDA kernels. Finally, Zhang et al. [38] propose an approach that uses nearest neighbors to construct the local features of each particle. Those local features are then averaged onto a regular grid. Like GNS, this method suffers from the need to repeat the neighbor search at every simulation timestep. Ultimately, the performance of point cloud-based simulators is tightly linked to the method used to process the spatial structure of the cloud. Brute force neighbor search is  $\mathcal{O}(N^2)$ , K-d trees are  $\mathcal{O}(N \log N)$ , and voxelization and hashing are  $\mathcal{O}(N)$  [18, 39]. Recently, Alkin et al. [40], attempt to bridge the gap between Lagrangian and Eulerian representations by avoiding explicit grid- or particle-based latent structures, but necessitating extra networks to account for particle movement or empty volumes.

An approach different from data-driven modeling is hybrid models, where parts of a classical solver are replaced with learned components. For instance, Yin et al. [41] employ a neural network to learn unknown physics, which is then integrated into a simulator. Similarly, Li et al. [42] use a neural network to bypass computational bottlenecks in MPM simulators, while Ma et al. [43] learn general constitutive laws, allowing for one-shot trajectory learning. These approaches achieve impressive results by leveraging extensive physics knowledge, but this reliance also limits their applicability. Hybrid models may inherit both the strengths and weaknesses of classical and ML methods. Our approach falls within the data-driven domain, as the prior constraints on the learnable dynamics induced by the MPM-like algorithm are minimal and can be overcome with data (Table 2.1).

## 2.3 NEURALMPM

We consider a Lagrangian system evolving in time and defined by the positions  $p_i^t$  and velocities  $v_i^t$  of a set of  $N$  particles  $i = 1, \dots, N$ . We denote by  $P^t$  and  $V^t$  the set of positions and velocities of all particles at time  $t$  and with  $S^t = (P^t, V^t)$  the full state of the system. In a more complex setting, the state of the system can include other local properties, such as pressure or elastic stiffness of materials, and global properties, such as an external force. In this work, for simplicity, we let the network learn the relevant simulation parameters implicitly. The evolution of the particles is described by a function  $f$  mapping the current state of the system to its next state  $S^{t+1} = f(S^t)$ . Given a starting system  $S^0 = (P^0, V^0)$ , its full trajectory, or rollout, is denoted by  $S^{1:T}$ . Our goal is to build an emulator  $\hat{f}_\theta(\cdot)$  capable of predicting a full rollout  $\hat{f}_\theta^{1:T}(S^0)$  of  $T$  timesteps from the initial state  $S^0$ . Following MPM, NeuralMPM operates in four steps, described below, as illustrated in Figure 2.1 and in Alg 1.

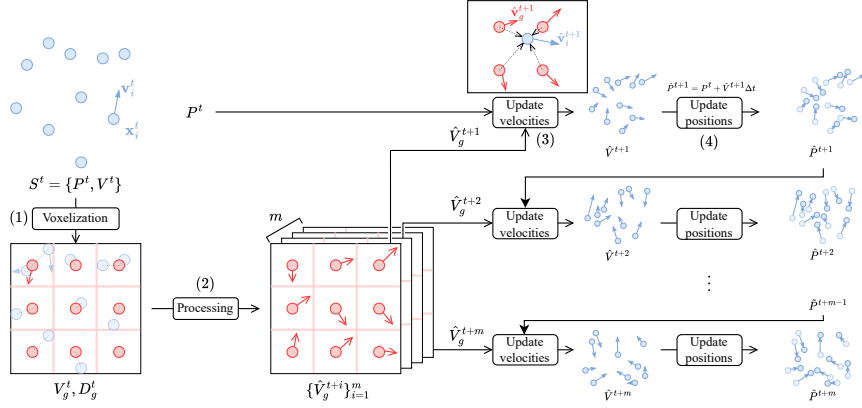


Figure 2.1. NeuralMPM works in 4 steps. (1) The positions  $P^t$  and velocities  $V^t$  of the particles are used to compute the velocity  $V_g^t$  and density  $D_g^t$  of each grid node through voxelization. (2) From this grid, the processor neural network predicts the grid velocities at the next  $m$  timesteps. The next  $m$  positions are computed iteratively by (3) performing bilinear interpolation of the predicted velocities onto the previous positions and (4) updating the positions using the predicted velocities.

**Step 1: Voxelization.** Using the particle positions  $P^t$ , the velocities  $V^t$  are interpolated onto a regular fixed-size grid. This interpolation is performed through *voxelization*, which divides the domain into regular volumes (voxels). Each grid node is identified as the center of a voxel (e.g., square in 2D) in the domain, and the velocities of the particles in the voxel are averaged to give the node’s velocity. Similarly, the density is computed as the normalized number of particles in the voxel. This results in the grid tensor  $G^t$  that contains the grid velocities  $V_g^t$  and density  $D_g^t$ .

**Step 2: Processing.** Taking advantage of the regular grid representation of the cloud, the grid velocities  $\{\hat{V}_g^{t+i}\}_{i=1}^m$  of the next  $m$  timesteps are predicted using a neural network. We chose a U-Net [44] as it is a well-established image-to-image model, known to perform well in various tasks, including physical applications. The combination of kernels applied with different receptive fields (from smaller to larger) can allow the U-Net to efficiently extract both local and global information. Nonetheless, any grid-to-grid architecture could be used. We experiment with the FNO [45] architecture in Appendix 2.D and find it to underperform, leading us to keep the U-Net. A fully convolutional U-Net and an FNO have the additional advantage of being able to generalize to different domain shapes, a desirable property (Section 2.4.3).

**Step 3: Update of particle velocities.** The predicted velocities  $\hat{V}^{t+1}$  at the next timestep are then interpolated back to the particle level onto the positions  $P^t$  using bilinear interpolation. The velocity of each particle is computed as a weighted average of the four surrounding grid velocities, based on its Euclidean distance to each of them.

**Step 4: Update of particle positions.** Finally, the positions of the particles are updated with Euler integration using the next velocities and known current positions of the particles, that is  $\hat{P}^{t+1} = P^t + \Delta t \hat{V}^{t+1}$ . Steps 3 and 4 are performed  $m$  times to compute the next  $m$  positions from the set of grid velocities computed at step 2.

Additional features of the individual particles can be included in the grid tensor  $G^t$  by interpolating them in the same way as the velocities. Local, such as boundary conditions, or global, such as gravity or external forces, features are represented as grid channels. For

simulations with multiple types of particles, the features of each material are interpolated independently and stacked as channels in  $G_t$ .

NeuralMPM is trained end-to-end on a set of trajectories  $S^{0:T}$  to minimize the mean squared error  $\|P^{t+1} - \hat{P}_\theta^{t+1}(S^t)\|_2^2$  between the ground-truth and predicted next positions of the particles. At inference time, the model is exposed to much longer sequences, which requires carefully stabilizing the rollout procedure to prevent the accumulation of large errors over time. To address this, we first make use of autoregressive training [14, 15], where the model is unrolled  $K$  times on its own predictions, producing a sequence of  $\hat{S}^k = \hat{f}_\theta(\hat{S}^{k-1})$  for  $k = 1, \dots, K$  and initial input  $\hat{S}^0 = S^0$ , before backpropagating the error through the entire rollout. Unlike more costly methods that require alternative stabilization strategies, such as noise injection [13], NeuralMPM’s efficiency makes autoregressive training possible. Nevertheless, to further stabilize the training, we couple autoregressive training with time bundling [32], resulting in a training strategy where the model predicts  $m$  steps  $\hat{S}^{1:m}$  at once from a single initial state, inside an outer autoregressive loop of  $K$  steps of length  $m$ . We show in Section 2.4 that this training strategy leads to more accurate rollouts.

---

**Algorithm 1** NeuralMPM

---

**Require:**  $P^0, V^0$ , grid  $g$ ,  $t$ , neural backbone  $h$ , functions **p2g** and **g2p**

- 1  $\hat{P}^t \leftarrow P^t, \hat{V}^t \leftarrow V^t$
- 2 **while**  $t < T$  **do**
- 3   **Voxelization:** Interpolate density and velocities onto the grid using  $G^t = (D_g^t, V_g^t) = \mathbf{p2g}(\hat{P}^t, \hat{V}^t)$
- 4   **Processing:**  $\{\hat{V}_g^i\}_{i=t+1}^{t+m} = h(G^t)$
- 5   **for**  $i$  in  $\text{range}(1, m)$  **do**
- 6     **Update velocities:**  $\hat{V}^{t+i} = \mathbf{g2p}(\hat{V}_g^{t+i}, \hat{P}^{t+i-1})$
- 7     **Update positions:**  $\hat{P}^{t+i} = P^{t+i-1} + \Delta t \hat{V}^{t+i}$
- 8    $t \leftarrow t + m$
- 9 **return Full trajectory**  $\{\hat{P}, \hat{V}\}_t^T$

---

## 2.4 EXPERIMENTS

We conduct a series of experiments to demonstrate the accuracy, speed, and generalization capabilities of NeuralMPM. Specifically, we examine its robustness to hyperparameter and architectural choices through an ablation study (2.4.1). We compare NeuralMPM to GNS and DMCF in terms of accuracy, training time, convergence, and inference speed (2.4.2). We also evaluate the generalization capabilities of NeuralMPM (2.4.3) and illustrate how its differentiability can be leveraged to solve an inverse design problem (2.4.4). Through these experiments, we demonstrate that NeuralMPM is a flexible, accurate, and fast method for emulating complex particle-based simulations. The baselines established by Winchenbach et al. [37] and hybrid simulators [42, 43] have promising results. However, we do not compare against them as they either use different benchmarks or are specifically tailored for certain physical domains, requiring material-specific knowledge. In contrast, NeuralMPM, like GNS and DMCF, requires only particle positions without being restricted to any particular domain. Rollouts for all experiments, models, and datasets are available at [drive](#) and in the supplementary material.

**Data.** We consider 6 datasets with variable sequence lengths, numbers of particles, and materials. The first three datasets, WATERRAMPS, SANDRAMPS, and GOOP, contain a single material, water, sand, and goop, respectively, with different material properties. The



first two datasets contain random ramp obstacles to challenge the model’s generalization capacity. The fourth dataset, MULTIMATERIAL, mixes the three materials together in the same simulations. These four datasets are taken from [13] and were simulated using the Taichi-MPM simulator [46]. They each contain 1000 trajectories for training and 30 (GOOP) or 100 (WATERRAMPS, SANDRAMPS, MULTIMATERIAL) for validation and testing. The fifth dataset, DAM BREAK 2D, was generated using SPH and contains 50 trajectories for learning, and 25 for validation and testing. The last dataset, VARIABLEGRAVITY, was also generated using Taichi-MPM. It consists of simulations with variable gravity of a water-like material, and contains 1000 trajectories for training and 100 for validation and testing.

**Protocol.** NeuralMPM is trained on trajectories with varying initial conditions and number of particles. The training batches are sampled randomly in time and across sequences. We use Adam [47] with the following learning rate schedule: a linear warm-up over 100 steps from  $10^{-5}$  to  $10^{-3}$ , 900 steps at  $10^{-3}$ , then a cosine annealing [48] for 100,000 iterations. We use a batch size of 128,  $K = 4$  autoregressive steps per iteration, bundle  $m = 8$  timesteps per model call (resulting in 24 predicted states), and a grid size of  $64 \times 64$ . For most of our experiments, we use a U-Net [44] with three downsampling blocks with a factor of 2, 64 hidden channels, a kernel size of 3, and MLPs with three hidden layers of size 64 for pixel-wise encoding and decoding into a latent space. For a fair comparison, we ran training and inference for NeuralMPM, DMCF, and GNS on the exact same hardware. GNS and DMCF were trained until convergence (a maximum of 240 and 120 hours, respectively), while NeuralMPM required 20 hours or less to converge. For WATERRAMPS, SANDRAMPS, GOOP, and MULTIMATERIAL, we use the same parameters as those reported by the authors. We perform a hyperparameter search for DMCF on DAM BREAK 2D and for both GNS and DMCF on VARIABLEGRAVITY and report the best performance obtained for a budget of 60 GPU-days per dataset. Further details on training can be found in Appendix 2.B.

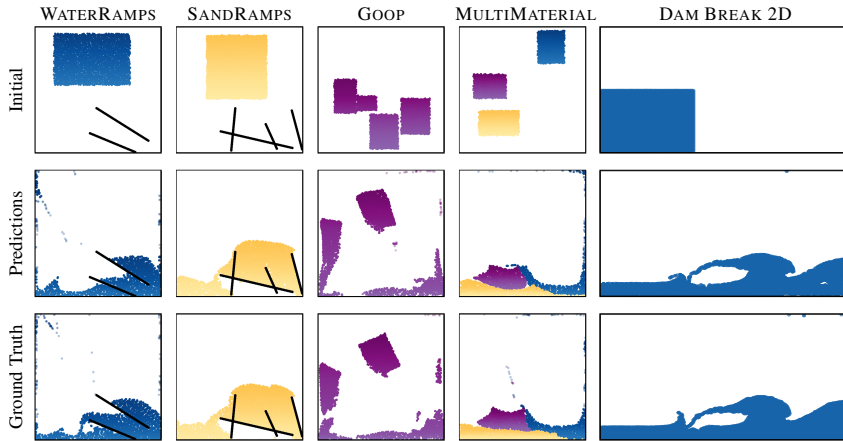


Figure 2.2. **Example snapshots.** We train and evaluate NeuralMPM on WATERRAMPS, SANDRAMPS and GOOP, each consisting of a single material, on MULTIMATERIAL that mixes water, sand and goop, and on DAM BREAK 2D, a rectangular-shaped SPH dataset. NeuralMPM is able to learn various kinds of materials, their interactions, and their interactions with solid obstacles. Despite being inspired by MPM, it is not limited to data showing MPM-like behavior.



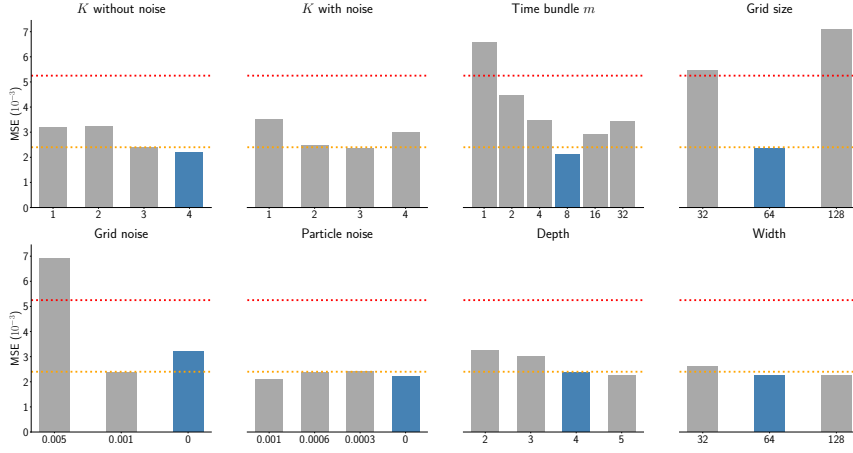


Figure 2.3. **Ablation results.** Mean squared error of full rollouts on unseen test data for Goop. The default parameters are in blue. The dotted orange line ( $2.4 \times 10^{-3}$ ) indicates the MSE we obtained for GNS after 240 hours (20M training steps). The dotted red line is the MSE for DMCF after the same amount of time ( $5.25 \times 10^{-3}$ ). NeuralMPM is robust to hyperparameter changes, with the biggest effects coming from the number of timesteps bundled together ( $m$ ) and grid noise. For a rollout of length  $T$ , the model is called  $T/m$  times, meaning lower values of  $m$  require maintaining stability for longer. Autoregressive training coupled with time bundling suffices to stabilize the model, eliminating the need for noise injection. Although GNS reportedly slightly outperforms NeuralMPM, these results could not be reproduced in our experiments.

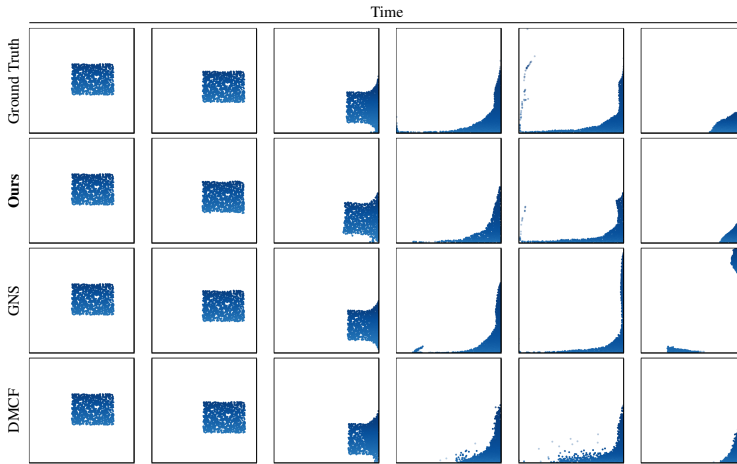


Figure 2.4. **Example VARIABLEGRAVITY trajectory against baselines.** Each method is unrolled starting from the initial conditions of a random test trajectory not seen during training.

### 2.4.1 ABLATION STUDY

To study the robustness of NeuralMPM to hyperparameter and architectural choices, we start with the default architecture and hyperparameters and ablate its components individually to examine their impact on performance. We vary the number  $K$  of autoregressive steps with and without grid and particle noise in the input state, the number of bundled timesteps  $m$  predicted by a single model call, and the depth and number of hidden channels of the network. We also investigate adding noise to stabilize rollouts, either directly to the particles’ positions or to the grid-level representation after voxelization.

Figure 2.3 summarizes the ablation results. A larger number  $K$  of autoregressive steps yields more accurate rollouts without the need to add noise. Indeed, injecting noise does not improve accuracy and is even detrimental for  $K = 4$ . Individually tuning the noise levels for grids and particles can modestly lower error rates, but is either very sensitive or negligible. The model performs better when bundling more timesteps, enabling faster rollouts as a single forward pass predicts more steps. We found  $m = 8$  to be optimal with the other default hyperparameters, outperforming larger bundling. This is because more network capacity is needed to extract information for the next 16 or 32 timesteps from a single state. Instead, we opted for a shallower and narrower network to balance speed and memory footprint with performance gains. In terms of network architecture, we chose a U-Net. We experiment with an FNO [45] in Appendix 2.D and find it to underperform, leading us to keep the U-Net architecture. We find the U-Net’s width and depth to have a minor impact on performance, confirming that a larger network is not needed. The grid size, however, is critical. A low resolution loses fine details, while a high resolution turns meaningful structures, such as liquid blobs or walls, into isolated voxels.

### 2.4.2 COMPARISON WITH PREVIOUS WORK

We compare NeuralMPM against GNS and DMCF. We use the official implementations and training instructions to assess training times, inference times, as well as accuracy. We compare against both GNS and DMCF on WATERRAMPS, SANDRAMPS, GOOP, DAM BREAK 2D, and VARIABLEGRAVITY. We also compare against GNS on MULTIMATERIAL, but not against DMCF since it does not support multiple materials.

**Accuracy.** We report quantitative results comparing the long-term accuracy in Table 2.1 and show trajectories of NeuralMPM in Figure 2.2, as well as comparisons against baselines on WATERRAMPS in Figure 2.4. On the mono-material datasets WATERRAMPS, SANDRAMPS, and GOOP, NeuralMPM performs competitively with GNS and better than DMCF in terms of mean squared error (MSE). For MULTIMATERIAL, NeuralMPM reduces the MSE by almost half, which we attribute to it being a hybrid method, known to better handle interactions, mixing, and collisions between different materials. In DAM BREAK 2D, NeuralMPM outperforms both baselines, despite the data being simulated using SPH. Finally, NeuralMPM surpasses the performance of DMCF in VARIABLEGRAVITY, even though the latter accounts for gravity explicitly. In terms of Earth Mover’s Distance (EMD), NeuralMPM outperforms both baselines across all benchmarks, suggesting that NeuralMPM is better at capturing the spatial distribution of the particles.

**Training.** In Figure 2.5, we report the evolution of the mean squared error of full emulated rollouts on the held-out test set during training, for each method, along with predicted snapshots at increasing training durations. NeuralMPM converges significantly faster than both baselines while reaching lower error rates. Furthermore, the convergence of the training procedure and quality of the architecture can be assessed much earlier during training, effectively saving compute and enabling the development of more refined

| Data (Simulator)      | $N$  | $T$  | NeuralMPM    |            | GNS          |      | DMCF  |      |
|-----------------------|------|------|--------------|------------|--------------|------|-------|------|
|                       |      |      | MSE↓         | EMD↓       | MSE↓         | EMD↓ | MSE↓  | EMD↓ |
| WATERRAMPS (MPM)      | 2.3k | 600  | 13.92        | <b>68</b>  | <b>11.75</b> | 90   | 20.45 | 105  |
| SANDRAMPS (MPM)       | 3.3k | 400  | <b>2.69</b>  | <b>61</b>  | 3.11         | 84   | 6.22  | 91   |
| GOOP (MPM)            | 1.9k | 400  | <b>2.18</b>  | <b>55</b>  | 2.4          | 73   | 5.25  | 85   |
| MULTIMATERIAL (MPM)   | 2k   | 1000 | <b>9.6</b>   | <b>66</b>  | 14.79        | 105  | -     | -    |
| DAM BREAK 2D (SPH)    | 5k   | 401  | <b>29.07</b> | <b>348</b> | 87.04        | 384  | 74.77 | 381  |
| VARIABLEGRAVITY (MPM) | 600  | 1000 | <b>14.48</b> | <b>92</b>  | 134          | 350  | 28.77 | 97   |

Table 2.1. Full rollout MSE & EMD (both  $\times 10^{-3}$ ) for NeuralMPM and the baselines on each dataset, with the maximum number of particles  $N$  and sequence length  $T$ . Each method was trained until full convergence (NeuralMPM: 15h, GNS: 240h, DMCF: 120h), and the best model was used.

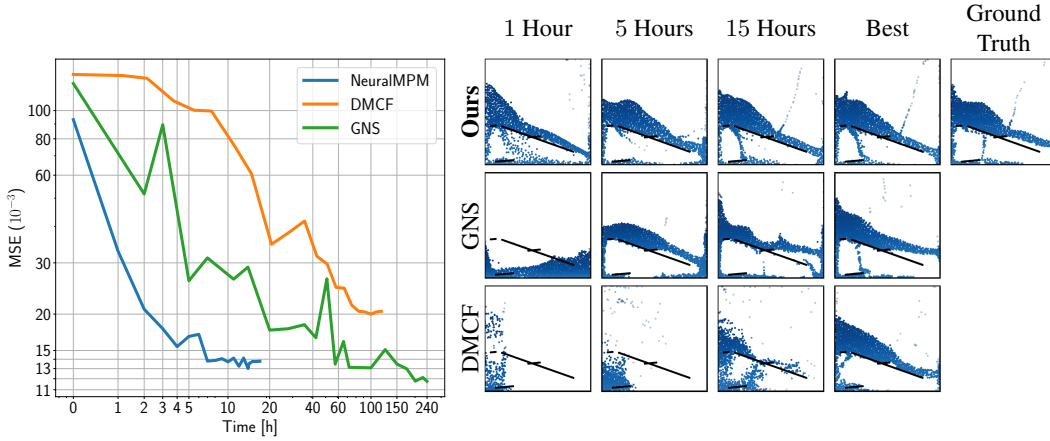


Figure 2.5. **Training convergence.** (Left) NeuralMPM trains and converges much faster than GNS and DMCF. Note the log scale on both axes. (Right) Snapshots of models trained for increasing durations then unrolled until the same timestep on a held-out simulation. For a fair comparison, out-of-bounds particles in GNS and DMCF were clamped.

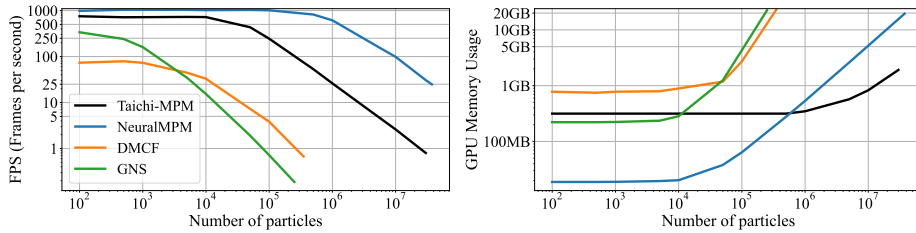


Figure 2.6. **Time and memory performance.** Average FPS (left) and GPU VRAM usage (right) for increasing numbers of particles for a traditional solver (Taichi-MPM [49]), NeuralMPM, and the two baselines. The two baselines quickly require very large amounts of memory and become very slow. Although Taichi-MPM is more memory efficient for high numbers of particles, NeuralMPM remains much faster, emulating 30 million particles at 25FPS. For the low particle count regime ( $< 10K$ ) we used the NeuralMPM and baselines WaterRamps models. For the high particle count regime we used untrained models and measured the throughput. The figure measures just FPS, and not the real simulation time. Taichi-MPM needs a much smaller step size than the three neural emulators (0.2ms vs 2.5ms), and is therefore likely slower than all of them.

final models. Moreover, NeuralMPM is also more memory-efficient, which enables the use of higher batch sizes of 128, as opposed to only 2 in GNS and DMCF.

**Inference time and memory.** In Figure 2.6, we display the time and memory performance of NeuralMPM, the two baselines GNS and DMCF, and the reference solver Taichi-MPM. In terms of speed, NeuralMPM strongly outperforms all three methods, partly thanks to time bundling, which considerably reduces the number of model calls required for a given number of frames to emulate. In terms of memory, although NeuralMPM remains inferior to Taichi-MPM, which is highly optimized, it can emulate tens of millions of particles on a single GPU, while GNS and DMCF struggle to reach half a million.

### 2.4.3 GENERALIZATION

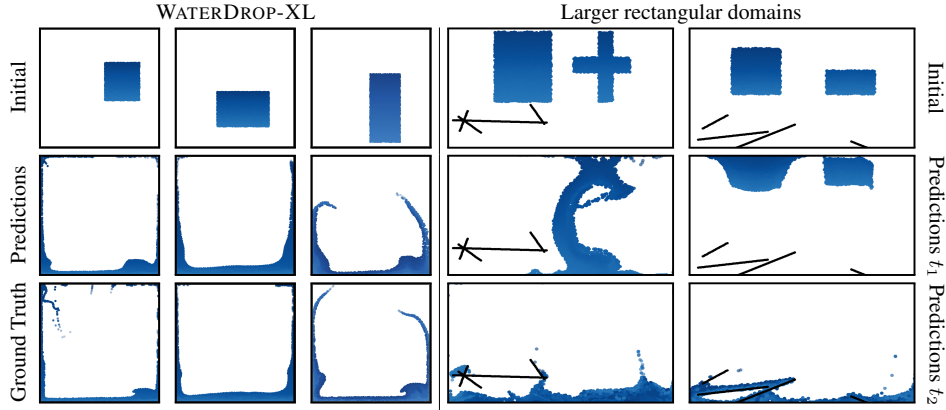


Figure 2.7. **Generalization.** (Left) NeuralMPM generalizes to domains with more particles ( $\sim 4\times$  here) with minimal inference time overhead due to the processing of the voxelized representation. (Right) A NeuralMPM model trained on a square domain can naturally generalize to larger rectangular domains (twice as wide here) when using a fully convolutional U-Net.

One notable advantage of NeuralMPM is that the processor is invariant to the number of particles, as the transition model only processes the voxelized representation, while both **p2g** and **g2p** scale linearly. To demonstrate this, we train a model on WATERRAMPS, which contains about 2.3k particles and 600 timesteps, and evaluate it on WATERDROP-XL, which features about four times more particles, 1000 timesteps, and no obstacles. An example snapshot is displayed in Figure 2.7. The larger number of particles only affects interpolation steps between the grid and particles, resulting in a negligible impact on total inference time, making the model nearly as fast despite 4 times more particles. We also validate generalization quantitatively by comparing the error rates on WATERDROP-XL of a model trained directly on it and the model trained solely on WATERRAMPS. With the same training budget, the latter achieves a lower MSE at  $20.92 \times 10^{-3}$  against  $28.09 \times 10^{-3}$ . More trajectories are displayed in Figure 2.23.

If a domain-agnostic processor architecture is used, such as a fully convolutional U-Net or an FNO, then NeuralMPM can generalize to different domain shapes without retraining, as shown in Fig 2.7. We demonstrate this ability by considering a model solely trained on WATERRAMPS, a square domain of size  $0.84 \times 0.84$  mapped to  $64 \times 64$  grids. Without retraining, we perform inference with this model on larger unseen environments of size  $1.68 \times 0.84$ , and change the grid size to  $128 \times 64$ . The unseen environments were built by

merging and modifying initial conditions of held-out test trajectories from WATERRAMPS. NeuralMPM emulates particles in this larger and rectangular domain despite being trained on a smaller square domain with a smaller grid, showing that a U-Net can generalize to other domains. No ground truth is displayed as [13] provide no information about the data generation. More trajectories are shown in Figure 2.24.

#### 2.4.4 INVERSE DESIGN PROBLEM

Finally, we demonstrate the application of NeuralMPM for inverse problems on a toy inverse design task that consists of optimizing the direction of a ramp to make the particles reach a target location, similar to [5]. We place a blob of water at different starting locations, and we then place a ramp at some location, with a random initial angle  $\alpha$ . The goal is to spin the ramp by tuning  $\alpha$  in order to make the water end up at a desired location. The main challenges of this task are the long-range time horizon of the goal and the presence of nonlinear physical dynamics. We proceed by selecting the point where we want the water to end up and compute the average distance between the point and particles at the last simulation frame. We then minimize the distance via gradient descent, leveraging the differentiability of NeuralMPM to solve this inverse design problem. We show an example optimization in Figure 2.8, and additional examples in Appendix 2.D.

*Everyone designs who devises courses of action aimed at changing existing situations into preferred ones.*

— Simon, *The Sciences of the Artificial*, 1969

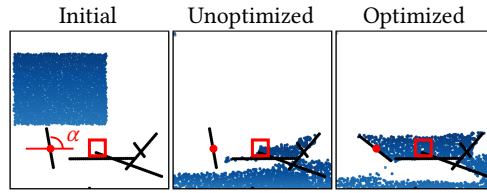


Figure 2.8. **Inverse design problem.** We exploit NeuralMPM’s differentiability to optimize the angle  $\alpha$  of a ramp, anchored at the red dot, in order to get the water close to the red square region.

## 2.5 CONCLUSION

**Summary.** We presented NeuralMPM, a neural emulation framework for particle-based simulations inspired by the hybrid Eulerian-Lagrangian Material Point Method. We have shown its effectiveness in simulating a variety of materials and interactions, its ability to generalize to larger systems and its use in inverse problems. Crucially, NeuralMPM trains in 6% of the time it takes to train GNS and DMCF to comparable accuracy, and is 5x-10x faster at inference time. By interpolating particles onto a fixed-size grid, global information is distilled into a voxelized representation that is easier to learn and process with powerful image-to-image models. The use of voxelization allows NeuralMPM to bypass expensive graph constructions, and the interpolation leads to easier generalization to a larger number of particles and constant runtime. The lack of expensive graph construction and message passing also allows for more autoregressive steps and parallel rollouts.

**Limitations.** Like other approaches, NeuralMPM is limited by the computation used to process the structure of the point cloud. In our case, voxelization means we cannot deal with particles that lie outside of the domain and are limited to regular grids. Additionally, the size of the voxels is directly related to the number of particles within a given volume. If the voxels are too large, the model will fail to capture finer details. Conversely, if they are too small, the model may struggle due to insufficient local structure. Similarly,

performance can degrade in very sparse domains. Compressible fluids might also present challenges, though this requires further verification.

**Future work.** Our work is only a first step towards hybrid Eulerian-Lagrangian neural emulators, leaving many avenues for future research. Extending NeuralMPM to 3D systems is a natural continuation of this work. Future studies could also explore alternative particle-to-grid and grid-to-particle functions, like the non-uniform Fourier transform [50], or more sophisticated interpolation methods from classical MPM literature [17]. A less traditional direction is to make NeuralMPM probabilistic and encode richer distributional information about the particles in the grid nodes, instead of maintaining only a mean value. This could potentially improve NeuralMPM’s ability to resolve subgrid phenomena. Finally, advances in Lagrangian Particle Tracking [51] will eventually make it possible to create datasets from real-world data, enabling the training of NeuralMPM directly from data without the need for the costly design process of a numerical simulator.

## REFERENCES

- [1] Omer Rochman-Sharabi, Sacha Lewin, and Gilles Louppe. “A Neural Material Point Method for Particle-based Emulation”. In *Transactions on Machine Learning Research* (2025).
- [2] J.J. Monaghan. “Smoothed Particle Hydrodynamics and Its Diverse Applications”. In *Annual Review of Fluid Mechanics* 44, Volume 44, 2012 (2012), pp. 323–346.
- [3] Renato Vacondio et al. “Grand challenges for Smoothed Particle Hydrodynamics numerical schemes”. In *Computational Particle Mechanics* 8.3 (May 2021), pp. 575–588.
- [4] Steven J. Lind, Benedict D. Rogers, and Peter K. Stansby. “Review of smoothed particle hydrodynamics: towards converged Lagrangian flow modelling”. In *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 476.2241 (2020), p. 20190801.
- [5] Kelsey Allen et al. “Inverse Design for Fluid-Structure Interactions using Graph Network Simulators”. In *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 13759–13774.
- [6] Qingqing Zhao, David B Lindell, and Gordon Wetzstein. “Learning to Solve PDE-constrained Inverse Problems with Graph Networks”. In *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 26895–26910.
- [7] Antonio Elia Forte et al. “Inverse Design of Inflatable Soft Membranes Through Machine Learning”. In *Advanced Functional Materials* 32.16 (2022), p. 2111610.
- [8] Remi Lam et al. “Learning skillful medium-range global weather forecasting”. In *Science* 382.6677 (2023), pp. 1416–1421.
- [9] Tobias Pfaff et al. “Learning Mesh-Based Simulation with Graph Networks”. In *International Conference on Learning Representations*. 2021.
- [10] Pablo Lemos et al. “Rediscovering orbital mechanics with machine learning”. In *Machine Learning: Science and Technology* 4.4 (Oct. 2023), p. 045002.
- [11] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In *Nature* 596.7873 (Aug. 2021), pp. 583–589.
- [12] Siyu He et al. “Learning to predict the cosmological structure formation”. In *Proceedings of the National Academy of Sciences* 116.28 (June 2019), pp. 13825–13832.
- [13] Alvaro Sanchez-Gonzalez et al. “Learning to Simulate Complex Physics with Graph Networks”. In *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 8459–8468.
- [14] Benjamin Ummenhofer et al. “Lagrangian Fluid Simulation with Continuous Convolutions”. In *International Conference on Learning Representations*. 2020.
- [15] Lukas Prantl et al. “Guaranteed Conservation of Momentum for Learning Particle-based Fluid Dynamics”. In *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 6901–6913.
- [16] Deborah Sulsky, Zhen Chen, and Howard L. Schreyer. “A particle method for history-dependent materials”. In *Computer Methods in Applied Mechanics and Engineering* 118 (1993), pp. 179–196.

- [17] Vinh Phu Nguyen, Alban de Vaucorbeil, and Stéphane Bordas. “The Material Point Method: Theory, Implementations and Applications (Scientific Computation) 1st ed. 2023 Edition”. Feb. 2023.
- [18] Yusheng Xu, Xiaohua Tong, and Uwe Stilla. “Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry”. In *Automation in Construction* 126 (2021), p. 103675.
- [19] Milad Rakhsha, Christopher E. Kees, and Dan Negrut. “Lagrangian vs. Eulerian: An Analysis of Two Solution Methods for Free-Surface Flows and Fluid Solid Interaction Problems”. In *Fluids* 6.12 (2021).
- [20] Arieh Iserles. “A First Course in the Numerical Analysis of Differential Equations”. 2nd ed. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2008.
- [21] K. W. Morton and D. F. Mayers. “Numerical Solution of Partial Differential Equations: An Introduction”. 2nd ed. Cambridge University Press, 2005.
- [22] Stephen B. Pope. “Turbulent Flows”. Cambridge University Press, 2000.
- [23] R. A. Gingold and J. J. Monaghan. “Smoothed particle hydrodynamics: theory and application to non-spherical stars”. In *Monthly Notices of the Royal Astronomical Society* 181.3 (Dec. 1977), pp. 375–389.
- [24] Daniel J. Price. “Smoothed particle hydrodynamics and magnetohydrodynamics”. In *Journal of Computational Physics* 231.3 (2012). Special Issue: Computational Plasma Physics, pp. 759–794.
- [25] Irina Ionescu et al. “Computational simulation of penetrating trauma in biological soft tissues using the material point method”. en. In *Stud Health Technol Inform* 111 (2005), pp. 213–218.
- [26] Hongbing Lu et al. “Multiscale simulation from atomistic to continuum – coupling molecular dynamics (MD) with the material point method (MPM)”. In *Philosophical Magazine* 86 (2006), pp. 2971–2994.
- [27] Shaofan Li and Wing Kam Liu. “Meshfree and particle methods and their applications”. In *Applied Mechanics Reviews* 55.1 (Jan. 2002), pp. 1–34.
- [28] Allen R. York II, Deborah Sulsky, and Howard L. Schreyer. “Fluid–membrane interaction based on the material point method”. In *International Journal for Numerical Methods in Engineering* 48.6 (2000), pp. 901–924.
- [29] Nitin P. Daphalapurkar et al. “Simulation of dynamic crack growth using the generalized interpolation material point (GIMP) method”. In *International Journal of Fracture* 143.1 (Jan. 2007), pp. 79–102.
- [30] Marcelo Alejandro Llano Serna, Márcio Muniz-de Farias, and Hernán Eduardo Martínez-Carvajal. “Numerical modelling of Alto Verde landslide using the material point method”. In *DYNA* 82.194 (Nov. 2015), pp. 150–159.
- [31] Alexey Stomakhin et al. “A material point method for snow simulation”. In *ACM Trans. Graph.* 32.4 (July 2013).
- [32] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. “Message Passing Neural PDE Solvers”. In *International Conference on Learning Representations*. 2022.
- [33] Bjoern List et al. “How Temporal Unrolling Supports Neural Physics Simulators”. 2024.
- [34] Omer Rochman Sharabi and Gilles Louppe. “Trick or treat? Evaluating stability strategies in graph network-based simulators”. In 2023.



- [35] Jiaqi Han et al. “Learning physical dynamics with subequivariant graph neural networks”. In *Advances in Neural Information Processing Systems* 35 (2022), pp. 26256–26268.
- [36] Hrishikesh Viswanath et al. “Reduced-Order Neural Operators: Learning Lagrangian Dynamics on Highly Sparse Graphs”. 2024.
- [37] Rene Winchenbach and Nils Thuerey. “Symmetric Basis Convolutions for Learning Lagrangian Fluid Mechanics”. 2024.
- [38] Yalan Zhang et al. “FluidsNet: End-to-end learning for Lagrangian fluid simulation”. In *Expert Systems with Applications* 152 (2020), p. 113410.
- [39] Erin Hastings and Jaruwan Mesit. “Optimization of large-scale, real-time simulations by spatial hashing”. In (Jan. 2005).
- [40] Benedikt Alkin et al. “Universal Physics Transformers: A Framework For Efficiently Scaling Neural Operators”. 2024.
- [41] Yuan Yin et al. “Augmenting physical models with deep networks for complex dynamics forecasting”. In *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (Dec. 2021), p. 124012.
- [42] Jin Li et al. “MPMNet: A Data-Driven MPM Framework for Dynamic Fluid-Solid Interaction”. In *IEEE Transactions on Visualization and Computer Graphics* 30.8 (Aug. 2024), pp. 4694–4708.
- [43] Pingchuan Ma et al. “Learning Neural Constitutive Laws From Motion Observations for Generalizable PDE Dynamics”. 2023.
- [44] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In *Medical Image Computing and Computer-Assisted Intervention*. 2015.
- [45] Zongyi Li et al. “Fourier Neural Operator for Parametric Partial Differential Equations”. In *International Conference on Learning Representations*. 2021.
- [46] Yuanming Hu et al. “A moving least squares material point method with displacement discontinuity and two-way rigid body coupling”. In *ACM Trans. Graph.* 37.4 (July 2018).
- [47] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. 2017.
- [48] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic Gradient Descent with Warm Restarts”. In *International Conference on Learning Representations*. 2017.
- [49] Yuanming Hu et al. “A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-Way Rigid Body Coupling”. In *ACM Transactions on Graphics (TOG)* 37.4 (2018), p. 150.
- [50] J.A. Fessler and B.P. Sutton. “Nonuniform fast Fourier transforms using min-max interpolation”. In *IEEE Transactions on Signal Processing* 51.2 (2003), pp. 560–574.
- [51] Andreas Schröder and Daniel Schanz. “3D Lagrangian Particle Tracking in Fluid Mechanics”. In *Annual Review of Fluid Mechanics* 55.Volume 55, 2023 (2023), pp. 511–540.
- [52] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In *Advances in neural information processing systems* 32 (2019).
- [53] Matthias Fey and Jan Eric Lenssen. “Fast graph representation learning with PyTorch Geometric”. In *arXiv preprint arXiv:1903.02428* (2019).

- [54] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In *International conference on machine learning*. pmlr. 2015, pp. 448–456.
- [55] Zongyi Li et al. “Geometry-informed neural operator for large-scale 3d pdes”. In *Advances in Neural Information Processing Systems* 36 (2024).
- [56] Alistair White et al. “Stabilized Neural Differential Equations for Learning Dynamics with Explicit Constraints”. 2024.
- [57] Patrick Schnell and Nils Thuerey. “Stabilizing Backpropagation Through Time to Learn Complex Physics”. 2024.
- [58] Bjoern List et al. “Differentiability in unrolled training of neural physics simulators on transient dynamics”. In *Computer Methods in Applied Mechanics and Engineering* 433 (2025), p. 117441.
- [59] Artur Toshev et al. “Lagrangebench: A lagrangian fluid mechanics benchmarking suite”. In *Advances in Neural Information Processing Systems* 36 (2024).
- [60] Johannes Brandstetter et al. “Geometric and Physical Quantities improve E(3) Equivariant Message Passing”. In *International Conference on Learning Representations*. 2022.
- [61] Marco Cuturi et al. “Optimal Transport Tools (OTT): A JAX Toolbox for all things Wasserstein”. In *arXiv preprint arXiv:2201.12324* (2022).

## 2.A SUPPLEMENTARY MATERIAL

In addition to the supplementary material on the project page (<https://neuralmpm.isach.be>), we have uploaded videos, minimal code for reproduction, and baseline comparisons to an online cloud storage that can be found at <https://bit.ly/neuralmpm-suppmaterial>.

## 2.B TRAINING DETAILS

**Hardware.** We run all our experiments using the same hardware: 4 CPUs, 24GB of RAM, and an NVIDIA RTX A5000 GPU with 24GB of VRAM. For reproducing the results of DMCF, we kept the A5000 GPU but it required up to 96GB of RAM for training.

**Data Preprocessing.** Similar to [15], we slightly alter the original MPM datasets to add boundary particles, as the original data from [13] does not have them. We define the velocity at a timestep to be  $\mathbf{v}_t = \mathbf{v}_t - \mathbf{v}_{t-1}$ . We therefore skip the first step during training for which no velocity is available.

**Implementation.** Our implementation, training scripts, experiment configurations, and instructions for reproducing results are publicly available at <https://neuralmpm.isach.be>. We implement NeuralMPM using PyTorch [52], and use PyTorch Geometric [53] for implementing efficient particle-to-grid functions, more specifically from the Scatter and Cluster modules. For memory efficiency, we do not store all (up to) 1,000 training trajectories in memory, and rather use a buffer of about 16 trajectories over which several epochs are performed before loading a new buffer of random trajectories.

**Baselines.** We use the official implementations and training instructions of GNS [13] and DMCF [15] to reproduce their results and conduct new experiments. More specifically, we train GNS as instructed for 20M steps on all four datasets, using their

provided configuration. For DMCF, we follow their default configurations and train for 400K iterations for each dataset. In datasets that were not used by the original authors, VARIABLEGRAVITY and DAM BREAK 2D, we performed a hyperparameter search. GNS and DMCF both were trained for a total budget of 60 GPU-days per dataset, and the best performance was reported.

**Normalization.** We normalize the input of the model over each channel individually. We investigated computing the statistics across a buffer, resembling [54], and precomputing them on the whole training set and found no difference in performance. During inference, we use the precomputed statistics.

**Code.** The code, together with additional videos, is available at the project’s website <https://neuralmpm.isach.be>.

## 2.C ADDITIONAL DISCUSSION

**Boundary conditions.** Boundary conditions are represented as stationary particles. Like moving particles, their density and zero velocity are interpolated into channels, but their positions are enforced to remain fixed during rollout. This approach allows for flexible boundary shapes, such as the ramps in WATERRAMPS.

**Voxelization and interpolation.** The voxelization procedure used is implemented using CUDA by PyTorchCluster [53], **p2g** is implemented by us based on the voxelized representation, and **g2p** uses PyTorch’s bilinear interpolation (`grid_sample`) [52] based on the voxelized representation of the data. The cost of those operations is  $\mathcal{O}(N)$ , e.g., a naive implementation of voxelization assigns each point  $\mathbf{p} = (x, y, z)$  to a voxel index computed as:

$$\mathbf{v} = \left( \left\lfloor \frac{x}{s} \right\rfloor, \left\lfloor \frac{y}{s} \right\rfloor, \left\lfloor \frac{z}{s} \right\rfloor \right),$$

where  $s$  is the voxel size and  $\lfloor \cdot \rfloor$  denotes the floor function. Since this computation is performed in constant time for each of the  $N$  points, the overall complexity is  $\mathcal{O}(N)$ . Alternative interpolation methods could be considered, although quick experimentation with bicubic interpolation for grid-to-particles or different weighting schemes for voxelization did not yield much difference in our pipeline. Some methods, like GIOROM [36] and GINO [55], learn the interpolation through the use of graph-based neural networks, but this strongly increases computational costs due to the construction of a graph and the forward pass in the neural network.

**Limitations of voxelization.** One of the main issues of voxelization onto a fixed regular grid is that the domain is limited and must be rectangular, like for the material point method. While a rectangular domain can be defined with local boundary conditions over the irregular domain of interest, this will be a disadvantage for sparse systems, for which Lagrangian methods might be a better fit. Likewise, out-of-bounds particles can be handled either with a larger domain, which might be more computationally expensive, by clamping the positions, or simply dropping the out-of-bounds particles. These particular conditions are known disadvantages of MPM, and of some implementations of SPH, and are therefore inherited by NeuralMPM.

**Rollout stability.** Long-term stability during rollout of neural emulators is an open problem. As such, we decouple this problem from the design of the method or the architecture. There are commonly used tricks in the literature, such as noise injection, backpropagating through the autoregressive rollout, temporal bundling, having two models work in tandem (one to produce large timesteps and one to interpolate between them), using explicit constraints [56], or modifying the gradients [57]. [58] perform a comparative study of some of these methods.

## 2.D SUPPLEMENTARY RESULTS

**Additional results on DamBreak2D** We have also compared the accuracy and inference speed of NeuralMPM against a different implementation of GNS, and one of SEGNN, both provided by [59], in Tables 2.2 and 2.3. As in Table 2.1, NeuralMPM outperforms both baselines by a margin.

|             | MSE↓         | EMD↓        |
|-------------|--------------|-------------|
| <b>Ours</b> | <b>20.76</b> | <b>2.88</b> |
| GNS         | 114.40       | 224.1       |
| SEGNN       | 124.39       | 268.4       |

Table 2.2. Full-trajectory MSE ( $\times 10^{-3}$ ) and Sinkhorn distance (EMD) ( $\times 10^{-4}$ ) for NeuralMPM, GNS, and SEGNN [60] on the DAM BREAK 2D dataset from LagrangeBench. The two latter models are baselines provided by LagrangeBench.

|             | Single call ( $T = 1$ ) | Rollout ( $T = 401$ ) |
|-------------|-------------------------|-----------------------|
| <b>Ours</b> | <b>7.41</b>             | <b>193.50</b>         |
| GNS         | 20.46                   | 8,170.47              |
| SEGNN       | 46.04                   | 18,194.10             |

Table 2.3. Inference time (in ms) of NeuralMPM, GNS, and SEGNN [60] on the DAM BREAK 2D dataset from LagrangeBench. Times were averaged over all test trajectories. NeuralMPM predicts 16 frames in a single model call and still outperforms the two baselines per call, which further widens the gap for the total rollout time.

**Evaluation.** In Table 2.4, we report the numerical MSE rollout values that were reported in the bar plots depicted in Figure 2.3 for GOOP. Also, Figures 2.9 and 2.10 display the error when rolling out a model for each dataset, both in terms of MSE and EMD. For both metrics, the error starts low and slowly accumulates over time. For the EMD, we use the Sinkhorn algorithm provided by [61].

## 2.D SUPPLEMENTARY RESULTS

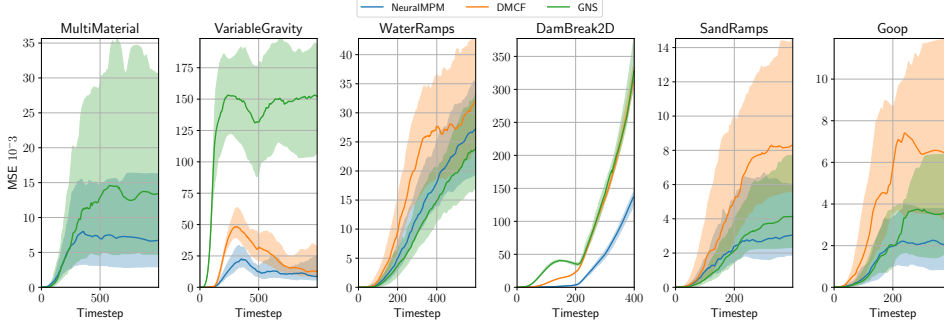


Figure 2.9. **MSE propagation during rollout.** We show the 25th, 50th and 75th percentile of the MSE, computed over particles and simulations, at each timestep during the rollout for each model and dataset. The accuracy decreases as errors accumulate. While training in 5% of the time, NeuralMPM outperforms the baselines on all datasets, except WaterRamps, where it is slightly worse than GNS.

| Parameter         | Value | MSE ( $\times 10^{-3}$ ) | Parameter      | Value  | MSE ( $\times 10^{-3}$ ) |
|-------------------|-------|--------------------------|----------------|--------|--------------------------|
| $K$ (No noise)    | 1     | 3.2                      | Grid noise     | 0      | 3.2                      |
|                   | 2     | 3.3                      |                | 0.001  | 2.4                      |
|                   | 3     | 2.4                      |                | 0.005  | 6.9                      |
|                   | 4     | 2.2                      |                | 0      | 2.2                      |
| $K$ (With noise)  | 1     | 3.5                      | Particle noise | 0.0003 | 2.4                      |
|                   | 2     | 2.5                      |                | 0.0006 | 2.4                      |
|                   | 3     | 2.4                      |                | 0.001  | 2.1                      |
|                   | 4     | 3.0                      |                | 2      | 3.3                      |
| Time bundling $m$ | 1     | 6.6                      | U-Net Depth    | 3      | 3.0                      |
|                   | 2     | 4.5                      |                | 4      | 2.4                      |
|                   | 4     | 3.5                      |                | 5      | 2.3                      |
|                   | 8     | 2.1                      |                | 32     | 2.6                      |
|                   | 16    | 2.9                      | U-Net Width    | 64     | 2.3                      |
|                   | 32    | 3.5                      |                | 128    | 2.2                      |
| Grid size         | 32    | 5.5                      |                |        |                          |
|                   | 64    | 2.4                      |                |        |                          |
|                   | 128   | 7.1                      |                |        |                          |

Table 2.4. Ablation results for Goop.

**Grid-to-grid network.** Although we have used a U-Net architecture for the grid-to-grid processor, NeuralMPM can be used with any grid-to-grid processor and is not limited to that network. For example, in Figure 2.13 and Table 2.5 we present qualitative and quantitative ablation results, respectively, for NeuralMPM using an FNO network [45] as the grid-to-grid processor. Results show that the FNO processor is slightly worse than the U-Net processor.

## 2.D SUPPLEMENTARY RESULTS

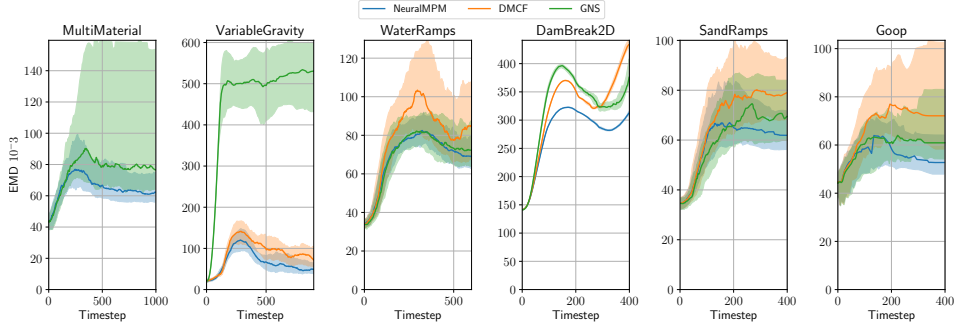


Figure 2.10. **EMD propagation during rollout.** We show the 25th, 50th and 75th percentile of the EMD, computed over particles and simulations, at each timestep during the rollout for each model and dataset. The accuracy decreases as errors accumulate. While training in 5% of the time, NeuralMPM outperforms the baselines on all datasets.

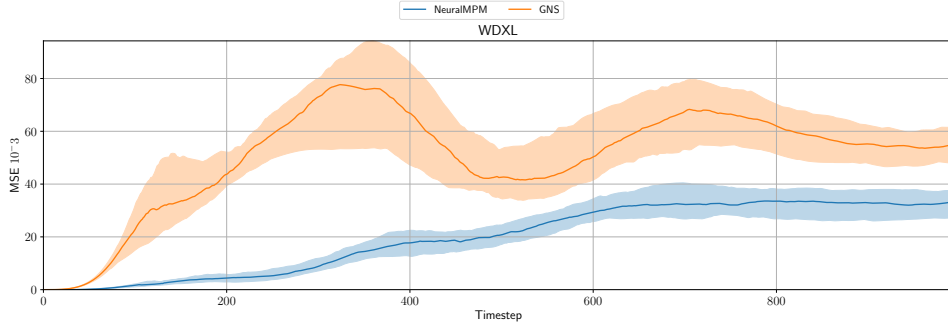


Figure 2.11. **MSE propagation during rollout for the generalization task WaterDrop-XL.** We show the 25th, 50th and 75th percentile of the MSE, computed over particles and simulations, at each timestep during the rollout for each model and dataset. The accuracy decreases as errors accumulate. The models used were trained on WaterRamps and tested on WaterDrop-XL to evaluate their generalization. NeuralMPM performs better despite having a slightly worse MSE on WaterRamps than GNS.

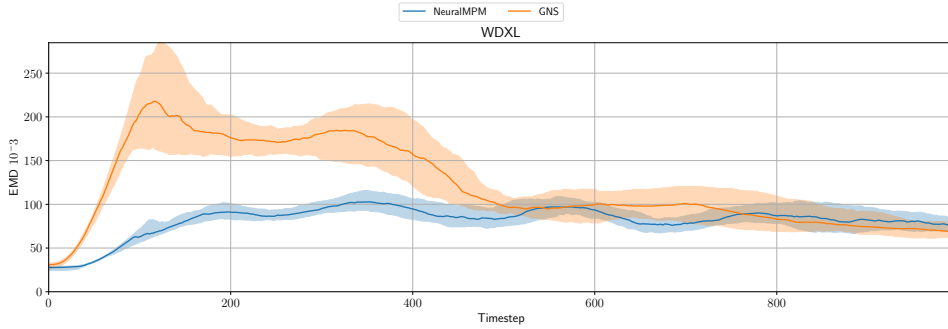


Figure 2.12. **EMD propagation during rollout for the generalization task WaterDrop-XL.** We show the 25th, 50th and 75th percentile of the EMD, computed over particles and simulations, at each timestep during the rollout for each model and dataset. The accuracy decreases as errors accumulate. The models used were trained on WaterRamps and tested on WaterDrop-XL to evaluate their generalization. NeuralMPM performs better despite having a slightly worse MSE on WaterRamps than GNS.

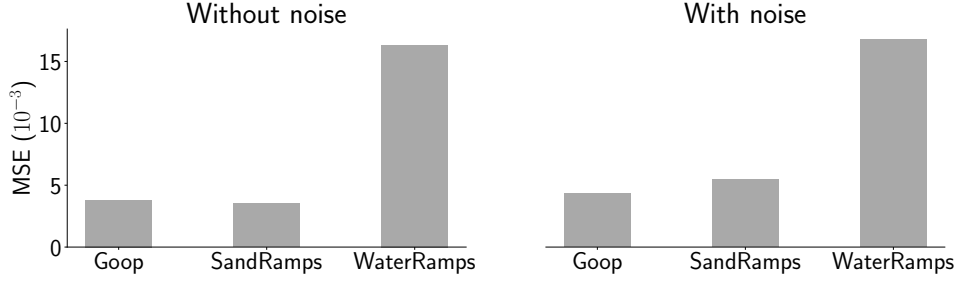


Figure 2.13. **FNO processor.** NeuralMPM with an FNO processor and default architecture. Rollout MSE ( $\times 10^{-3}$ ) for different datasets.

| Data       | FNO with noise | FNO without noise |
|------------|----------------|-------------------|
| WATERRAMPS | 16.8           | 16.3              |
| SANDRAMPS  | 5.5            | 3.5               |
| GOOP       | 4.3            | 3.8               |

Table 2.5. Rollout MSE ( $\times 10^{-3}$ ) for NeuralMPM with an FNO processor and default architecture, with and without noise.

**Additional inverse problem examples.** We show two additional optimization examples in Figure 2.14.

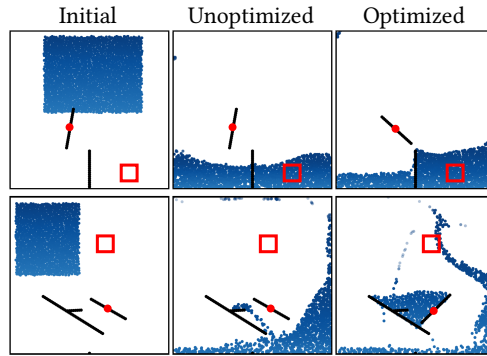


Figure 2.14. **Inverse design problem.** Additional optimization examples. We exploit NeuralMPM’s differentiability to optimize the angle  $\alpha$  of a ramp, anchored at the red dot, in order to get the water close to the red square region.

## 2.E GALLERY OF PREDICTED TRAJECTORIES

We present additional rollout comparisons in Figures 2.15 and 2.16. Further, in addition to the trajectories in Figures 2.2 and 2.4, we show additional trajectories emulated with NeuralMPM for all datasets in Figures 2.17, 2.18, 2.19, 2.20, 2.21, 2.22, 2.23, and 2.24. We also release *videos* in the supplementary material, which we recommend watching to better see the details and limitations of NeuralMPM. This includes 10 videos per dataset of emulated trajectories on held-out test simulations.

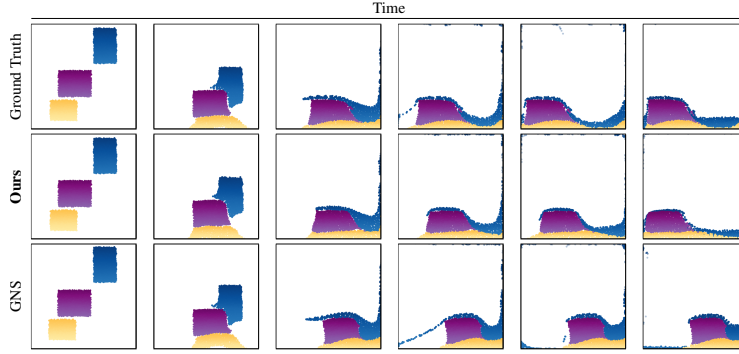


Figure 2.15. **Example MULTIMATERIAL trajectory against baselines.** Each method is unrolled starting from the initial conditions of a random test trajectory not seen during training.

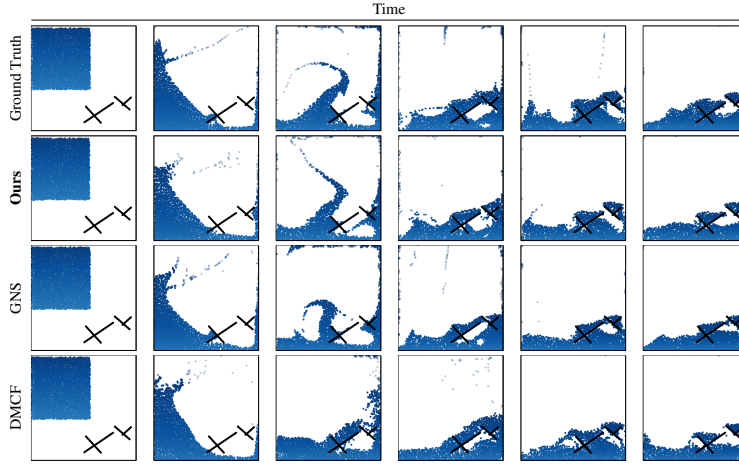


Figure 2.16. **Example WATERRAMPS trajectory against baselines.** Each method is unrolled starting from the initial conditions of a random test trajectory not seen during training.



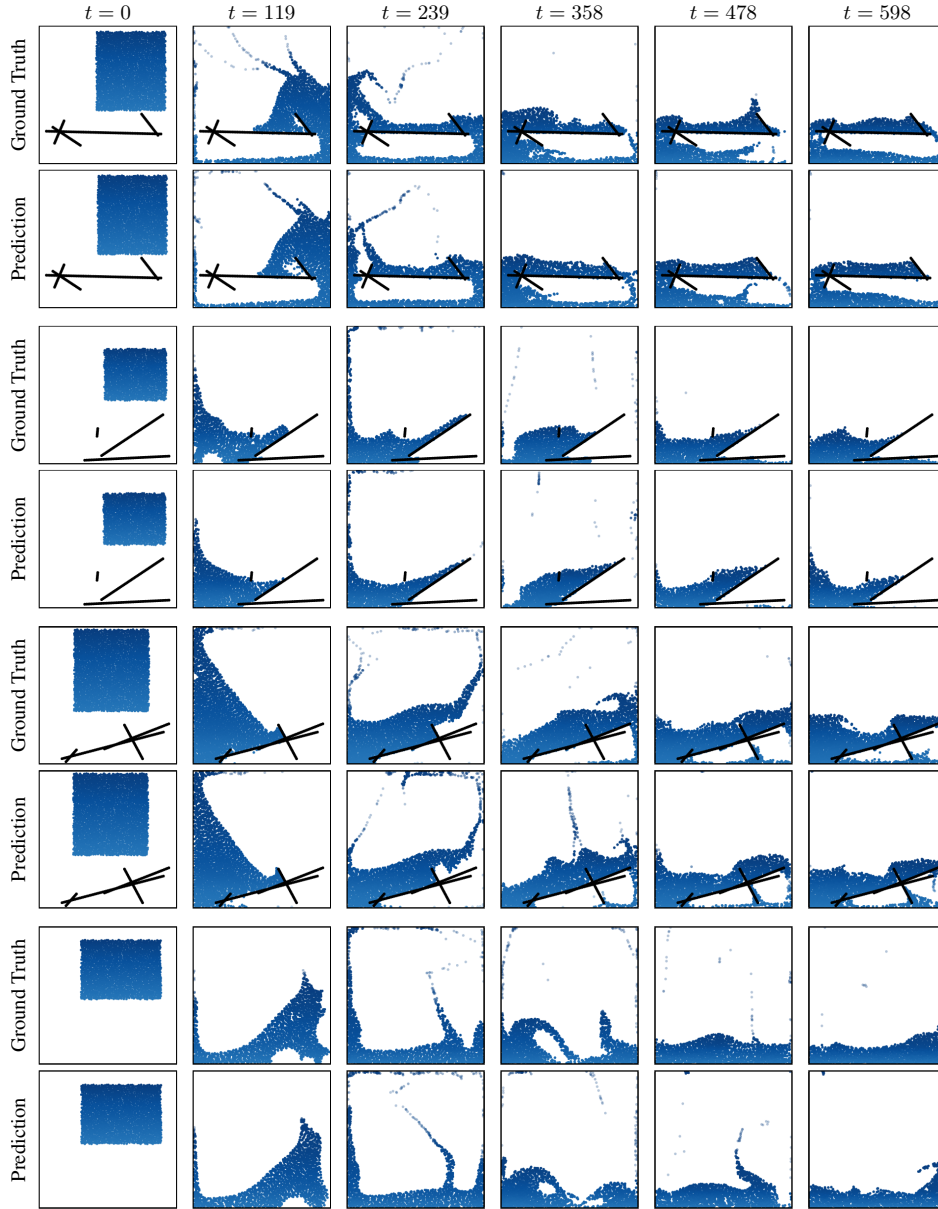


Figure 2.17. **Additional WATERRAMPS predicted trajectories.** Evenly spaced in time snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set.

## 2.E GALLERY OF PREDICTED TRAJECTORIES

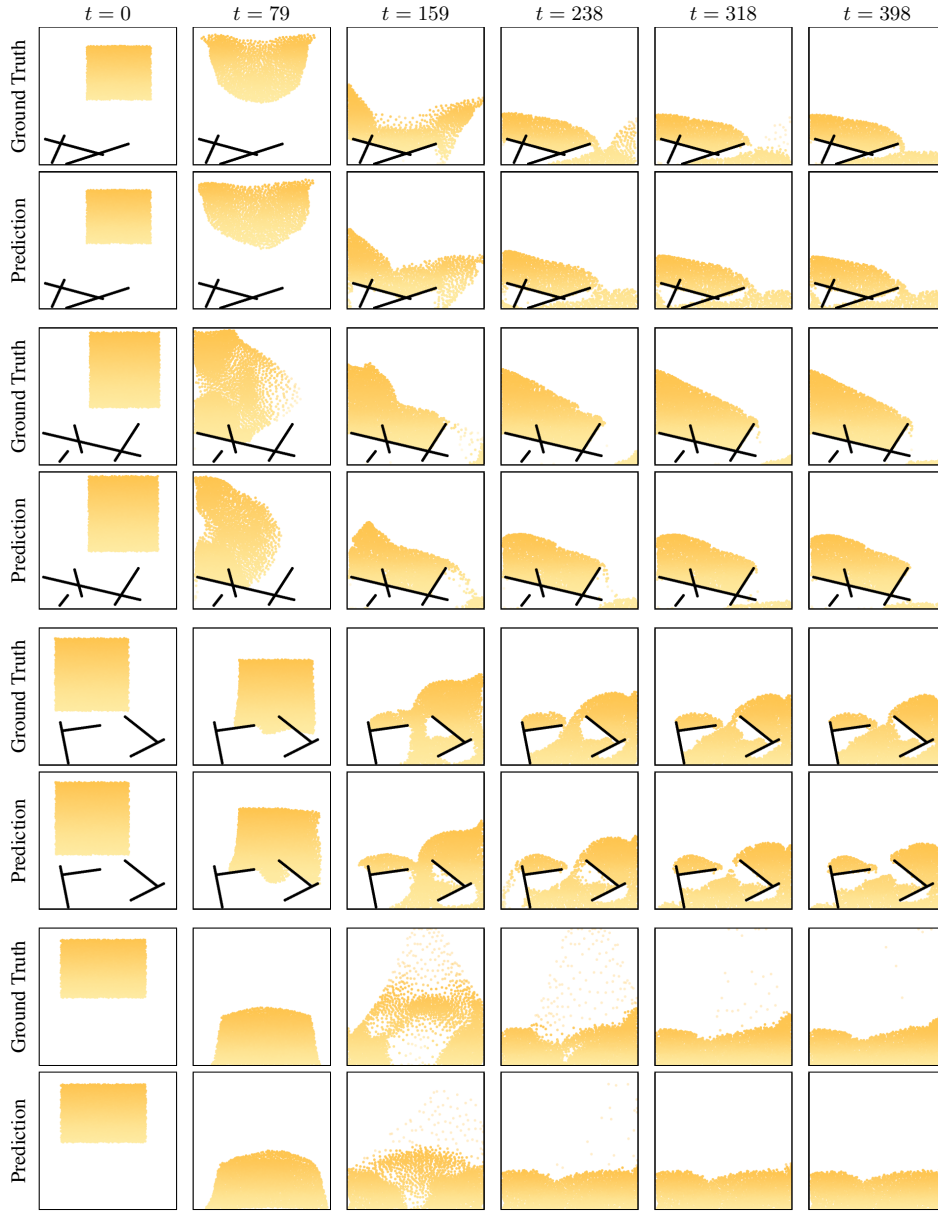


Figure 2.18. **Additional SANDRAMPS predicted trajectories.** Evenly spaced in time snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set.

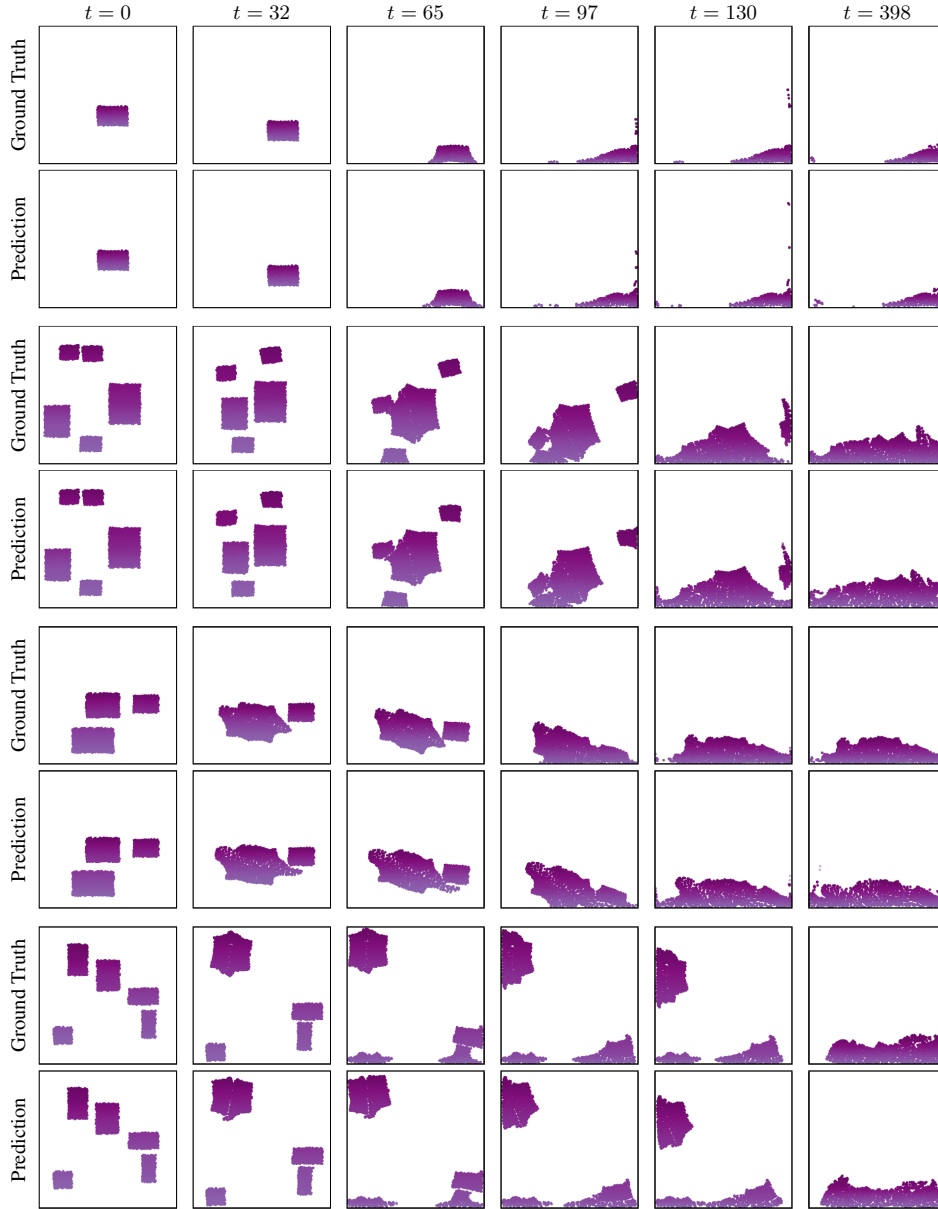


Figure 2.19. **Additional Goop predicted trajectories.** Snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set. Due to Goop quickly reaching equilibrium, more snapshots are taken in the first half of the trajectory.

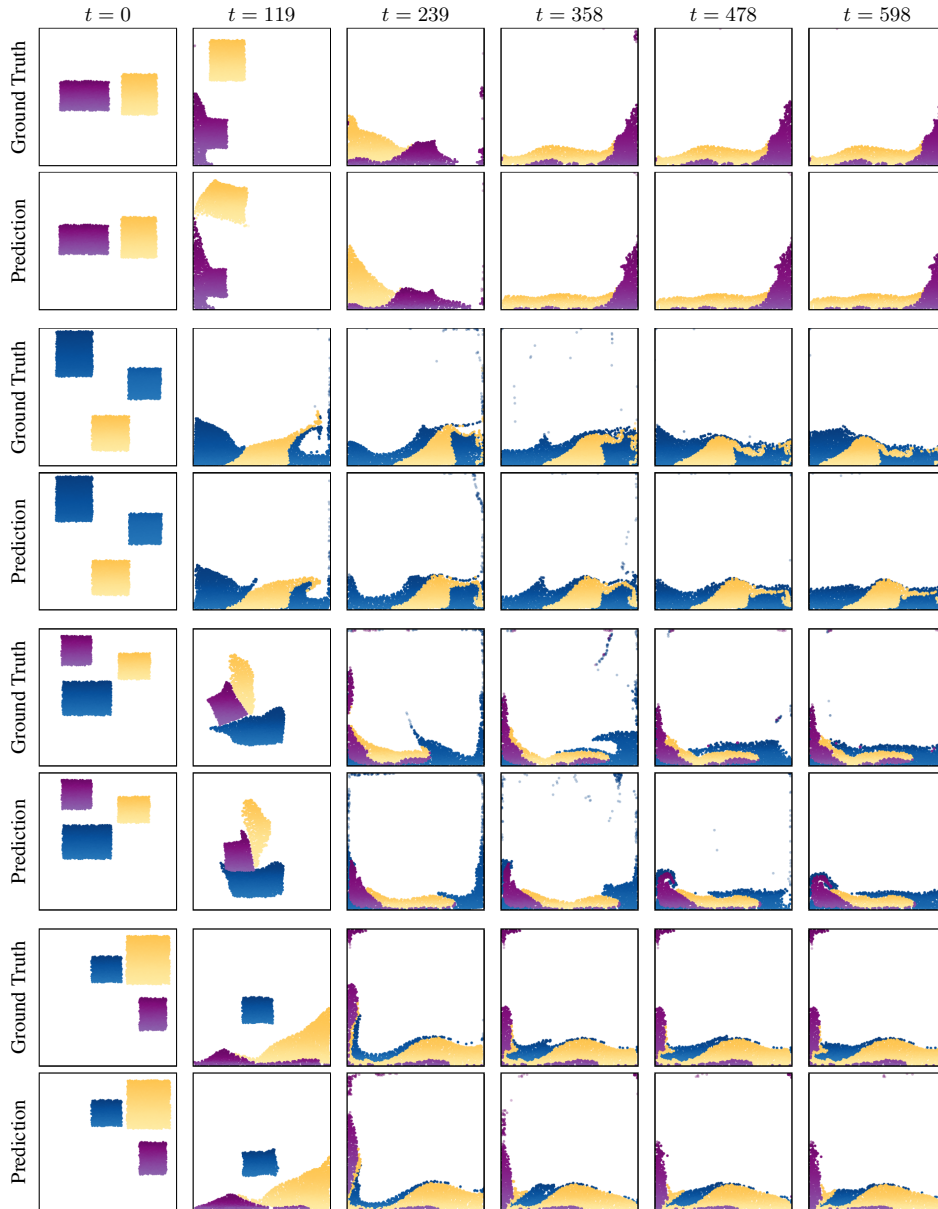


Figure 2.20. **Additional MULTIMATERIAL predicted trajectories.** Evenly spaced in time snapshots of predicted unrolled trajectories against ground truth. All trajectories are from the held-out test set. The first trajectory illustrates a rare failure where the shape of sand particles is not retained, even though all particles are supposed to maintain the same velocity while airborne, as they are thrown against the wall.

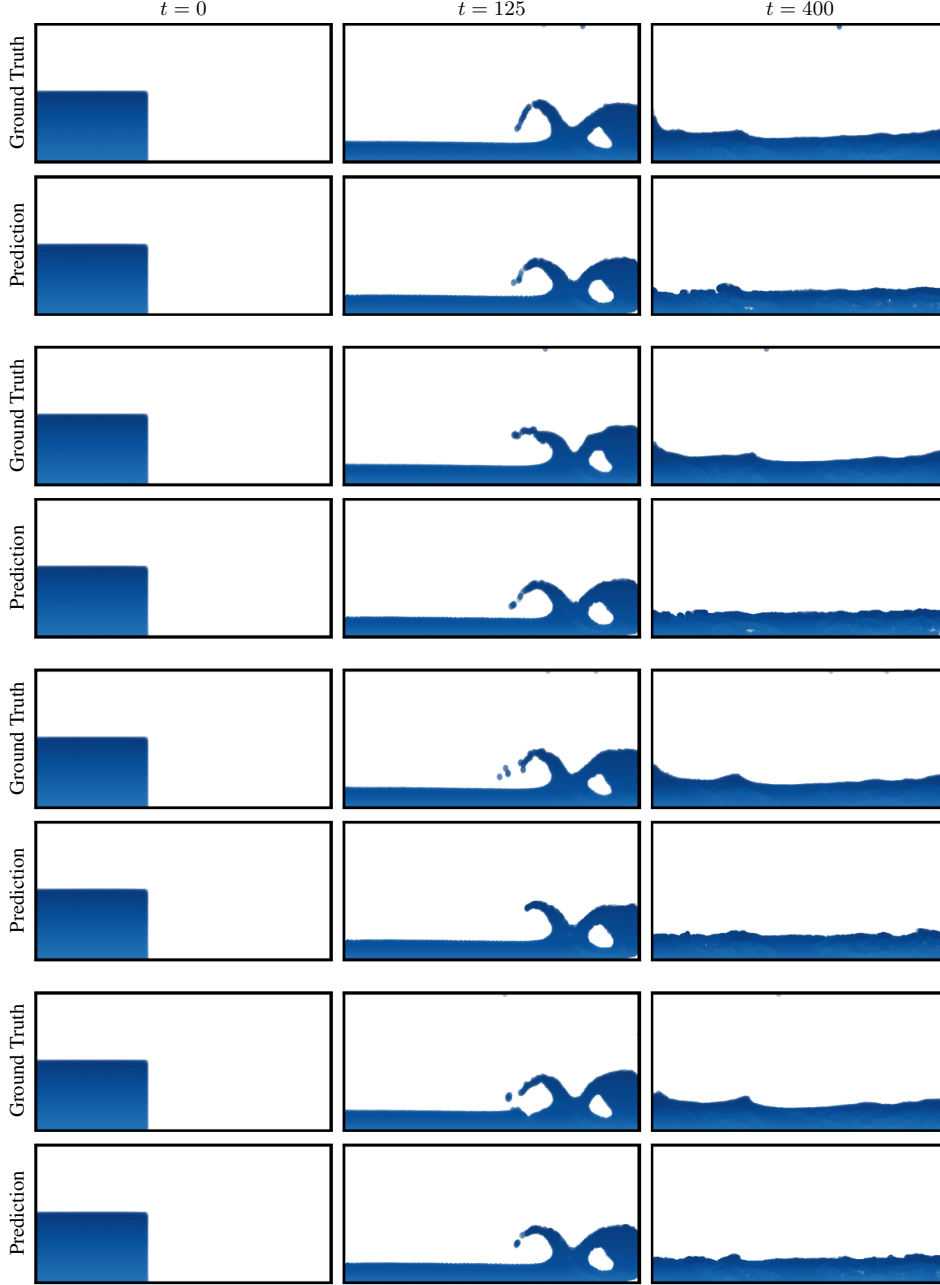


Figure 2.21. **Additional DAM BREAK 2D predicted trajectories.** Snapshots of predicted trajectories against ground truth. All trajectories come from the held-out test set. To better show the differences of these longer sequences, we select the following timesteps not even in time:  $t \in \{0, 125, 400\}$ . In the last trajectory, NeuralMPM struggles to follow the gravity direction and breaks down over time.

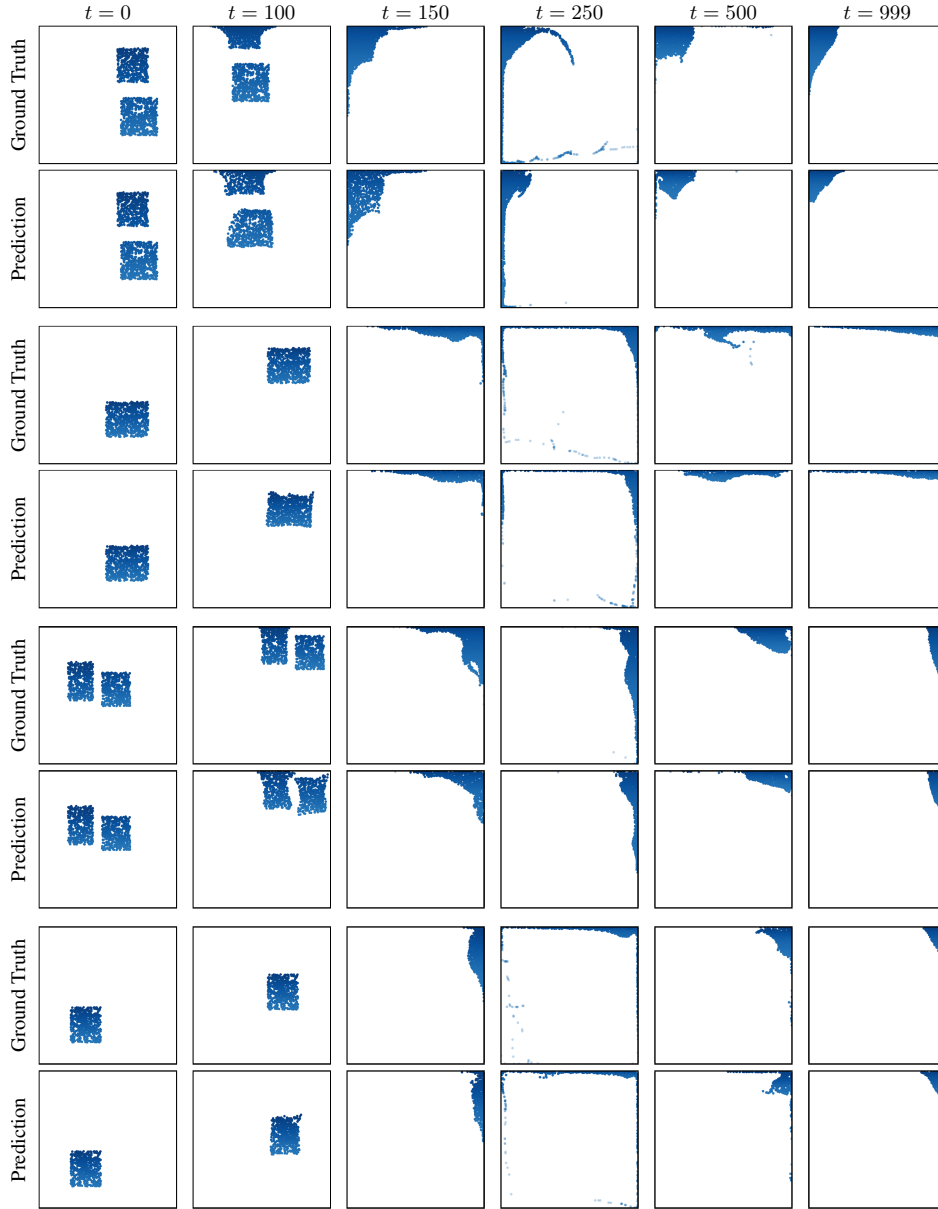


Figure 2.22. **Additional VARIABLEGRAVITY predicted trajectories.** Snapshots of predicted trajectories against ground truth. All trajectories come from the held-out test set.

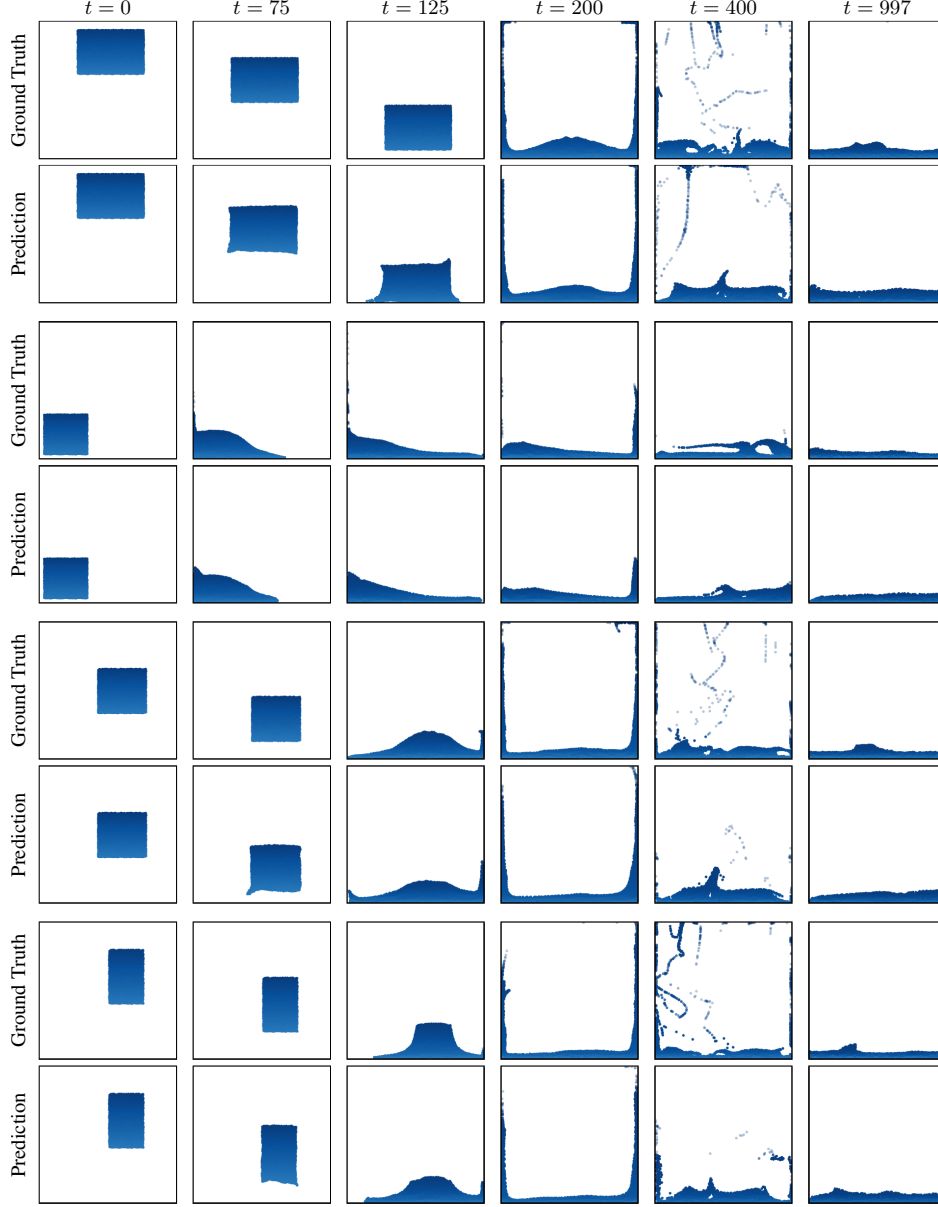


Figure 2.23. **Generalization to more particles on WATERDROP-XL.** Snapshots of predicted trajectories emulated using a model trained solely on WATERRAMPS, against ground truth. All trajectories come from the held-out test set of WATERDROP-XL. To better show the differences of these longer sequences, we select the following timesteps, which are not evenly spaced in time:  $t \in \{0, 75, 125, 200, 400, 997\}$ . We can observe that the generalizing model struggles to retain the shape of water while it's falling.

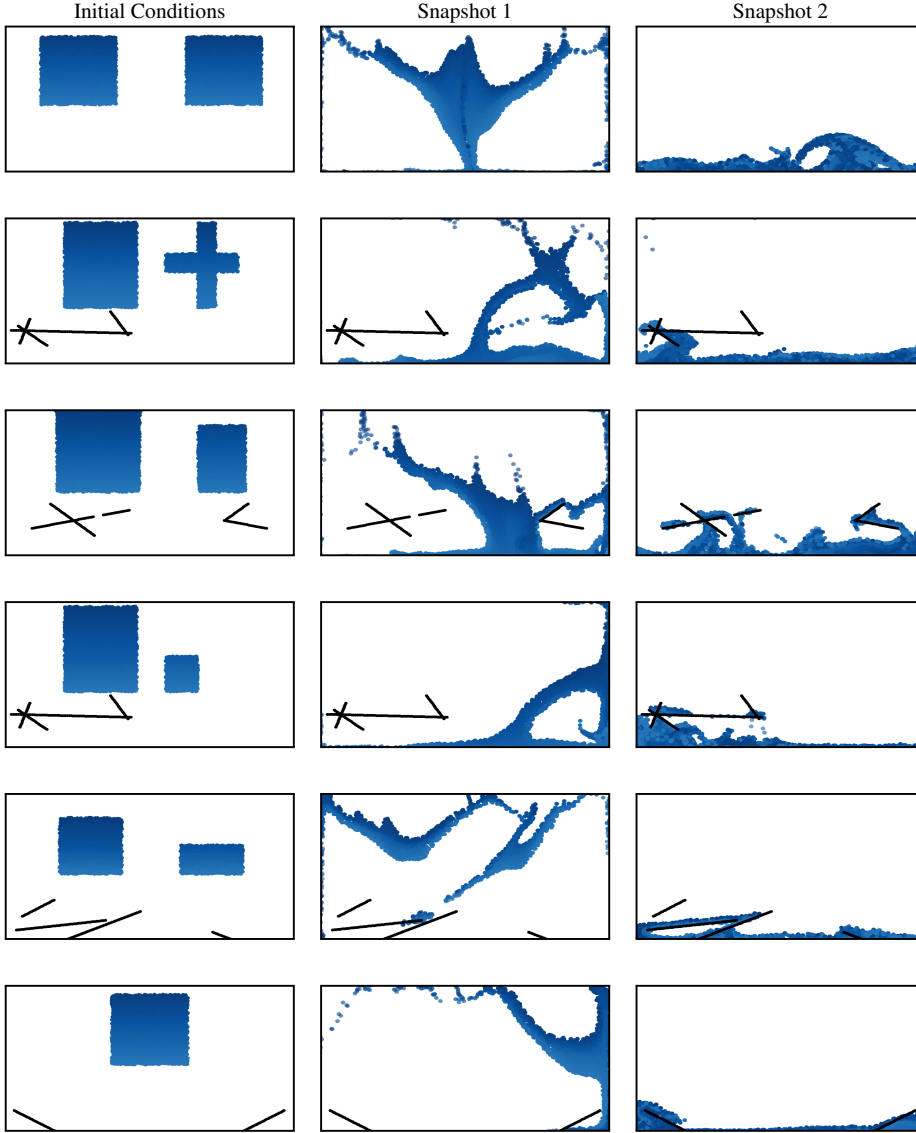


Figure 2.24. **Generalization to larger and non-square domains.** We train a model on the square domains in WATERRAMPS using  $64 \times 64$  input grids to the U-Net, and then perform inference on manually generated non-square environments that are twice as wide and use a  $128 \times 64$  input grid to the same U-Net. NeuralMPM flawlessly generalizes and emulates particles in these new environments. Note: no ground truth is available because the authors of GNS did not provide the physical parameters for simulating WATERRAMPS using Taichi. Chosen time steps are 0, 150, 575. We recommend watching the videos in the supplementary material for more detailed evaluation.



### 3 APPA: BENDING WEATHER DYNAMICS WITH LATENT DIFFUSION MODELS FOR GLOBAL DATA ASSIMILATION

*O God, I could be bounded in a nutshell and count myself a King of infinite space.*

— Shakespeare, *Hamlet*, II, 2

#### ADDENDUM

This chapter appeared previously as

G r me Andry, Fran ois Rozet, Sacha Lewin, Omer Rochman, Victor Mangeleer, Matthias Pirlet, Elise Faulx, Marilaure Gr goire, and Gilles Louppe. “Appa: Bending Weather Dynamics with Latent Diffusion Models for Global Data Assimilation”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025.

G r me and Sacha implemented the model, ran the experiments, interpreted the results, and wrote most of the manuscript. Fran ois proposed the methods and contributed to the codebase and to the text. Omer took part in the design of the architecture and of the observation models, iterated on the posterior sampling, and contributed to the codebase and to the writing. Victor and Matthias worked on the codebase and on the visualization of the results, and Elise provided domain knowledge. Gilles supervised the project, suggested experiments, and participated in the writing.

For the version presented in this chapter, we have edited the structure and content of the manuscript to be consistent with the rest of this document. This work was presented at the Machine Learning and the Physical Sciences Workshop at NeurIPS 2025. The Appa project is nevertheless still ongoing, and we wish to include it in the dissertation, as it is a logical continuation of our work.

#### ABSTRACT

Deep learning has advanced weather forecasting, but accurate predictions first require identifying the current state of the atmosphere from observational data. In this work, we introduce Appa, a score-based data assimilation model generating global atmospheric trajectories at 0.25  resolution and 1-hour intervals. Powered by a 565M latent diffusion model trained on ERA5, Appa can be conditioned on arbitrary observations to infer

plausible trajectories, without retraining. Our probabilistic framework handles reanalysis, filtering, and forecasting within a single model, producing physically consistent reconstructions from various inputs. Results establish latent score-based data assimilation as a promising foundation for future global atmospheric modeling systems.

### 3.1 INTRODUCTION

Data assimilation combines observational data with physical models to estimate atmospheric states. Formally, let  $x^{1:L} = (x^1, x^2, \dots, x^L) \in \mathbb{R}^{L \times V \times C}$  denote a trajectory of  $L$  atmospheric states, each represented as  $C$  physical fields over a mesh of  $V$  vertices. Let  $p(x^1)$  be the initial state prior and  $p(x^{i+1} | x^i)$  the transition dynamics. Observations  $y \in \mathbb{R}^M$  of the state trajectory  $x^{1:L}$  follow an observation process  $p(y | x^{1:L})$ , generally formulated as  $y = \mathcal{M}(x^{1:L}) + \eta$ , where the measurement function  $\mathcal{M} : \mathbb{R}^{L \times V \times C} \mapsto \mathbb{R}^M$  can be non-linear and  $\eta \in \mathbb{R}^M$  represents observational error that accounts for instrumental noise and systematic uncertainties. The goal of data assimilation is to infer plausible trajectories  $x^{1:L}$  consistent with the observations, that is, to estimate the trajectory posterior

$$p(x^{1:L} | y) = \frac{p(y | x^{1:L})}{p(y)} p(x^1) \prod_{i=1}^{L-1} p(x^{i+1} | x^i). \quad (3.1)$$

While Eq. (3.1) defines the posterior inference problem generically, assimilation tasks correspond to specific choices of states  $x^i$  to infer and observations  $y$  to take into account. In this work, we focus on three practical cases:

$$\textbf{Reanalysis} \quad x^{1:L} \sim p(x^{1:L} | y^{1:L}), \quad (3.2)$$

$$\textbf{Filtering} \quad x^L \sim p(x^L | y^{1:L}), \quad (3.3)$$

$$\textbf{Forecasting} \quad x^{K+1:L} \sim p(x^{K+1:L} | y^{1:K}) \quad \text{or} \quad p(x^{K+1:L} | x^{1:K}). \quad (3.4)$$

Reanalysis aims to reconstruct full trajectories from partial historical observations of the same time segment. The primary purpose of reanalysis is to create datasets of historical data for the land, atmosphere, and oceans. These datasets enable scientists to better monitor and understand the climate, conduct surveys, and develop new weather models. Filtering, in contrast, only infers the posterior distribution of the current state, obtained as a marginal of the reanalysis posterior. Forecasting, as its name suggests, extends beyond the observed segment, producing posterior distributions over future states. The forecasting problem is often initialized by an external estimation of the current state  $x^K$ , or decomposed into first estimating this current state from  $y^{1:K}$ , and then predicting its evolution.

Traditional methods like 4D-Var [2–6] and ensemble Kalman filters [7] are effective but rely on linearizations, require expensive differentiation, and provide point estimates rather than full posterior distributions [8]. Recent data-driven approaches [9–13] integrate deep learning into assimilation or forecast directly from observations, but suffer from limited resolution, lack of uncertainty quantification, and require retraining for new observation configurations.

### 3.2 APPA

Appa combines score-based data assimilation [14–16] with latent diffusion models for physics emulation [17], scaled to the global atmospheric system at  $0.25^\circ$  resolution and 1-hour intervals, with 6 surface fields and 5 atmospheric fields across 13 pressure levels.

*One has to know with sufficient accuracy the state of the atmosphere at a given time.*  
— Bjerknes, *The Problem of Weather Prediction*, 1904

**Architecture** Appa consists of a 340M encoder-decoder pair  $(E_\psi, D_\psi)$  that compresses atmospheric states  $x^i$  by a factor of 530 into latent representations  $z^i \sim \mathcal{N}(z^i | E_\psi(x^i), \sigma_z^2 I)$ . This reduces the dimensionality from  $\mathcal{O}(10^8)$  elements per atmospheric state ( $\mathcal{O}(10^{10})$  for 4 days at 1-hour resolution) to  $\mathcal{O}(10^5)$  elements per latent state ( $\mathcal{O}(10^7)$  for 4 days), enabling efficient generation and inference in the latent space. The encoder-decoder pair is trained with a latitude- and level-weighted mean squared error loss [18, 19].

The autoencoder is paired with a 225M diffusion transformer (DiT) [20] that operates on windows of  $W = 24$  consecutive latent states. Following [15, 21], we train a denoiser to estimate the denoising posterior mean  $\mathbb{E}[z^{i:i+W} | z_t^{i:i+W}]$  which, via Tweedie’s first-order formula [22–24], provides the prior score function  $\nabla_{z_t^{i:i+W}} \log p(z_t^{i:i+W})$  needed for the reverse diffusion sampling process. For a variance exploding diffusion process [25], we have

$$\mathbb{E}[z^{i:i+W} | z_t^{i:i+W}] = z_t^{i:i+W} + \sigma_t^2 \nabla_{z_t^{i:i+W}} \log p(z_t^{i:i+W}). \quad (3.5)$$

As in the SDA [14, 15] framework, the score over trajectories  $z_t^{i:i+W}$  is approximated by composing the scores over windows. We generalize SDA’s composition algorithm by introducing a time stride  $\Delta \geq 1$  between consecutive windows. Using a larger stride reduces the window overlap and, therefore, the number of network evaluations. Detailed algorithms for training and composing local scores can be found in Appendix 3.B.2.

**Sampling conditionally on weather observations** To sample from the posterior  $p(z^{1:L} | y)$ , we replace the prior score in the reverse diffusion sampling process with the posterior score

$$\nabla_{z_t^{1:L}} \log p(z_t^{1:L} | y) = \nabla_{z_t^{1:L}} \log p(z_t^{1:L}) + \nabla_{z_t^{1:L}} \log p(y | z_t^{1:L}). \quad (3.6)$$

The prior score  $\nabla_{z_t^{1:L}} \log p(z_t^{1:L})$  is obtained from the denoiser and the combination of scores over windows, while the likelihood score  $\nabla_{z_t^{1:L}} \log p(y | z_t^{1:L})$  can be approximated without retraining [14, 26–28] under moderate assumptions on the observation process.

The key challenge is that the observation process  $p(y | x^{1:L})$  is defined for atmospheric states  $x^i$  rather than latent states  $z^i$ . For observation sequences  $y^{1:L}$  of the form  $y^i = \mathcal{M}^i(x^i) + \eta^i$ , we approximate the mapping from  $z^i$  to  $y^i$  as the composition of the decoder  $D_\psi$  and measurement operator  $\mathcal{M}^i$ . Formally,

$$p(y^{1:L} | z^{1:L}) \approx \mathcal{N}(y^{1:L} | \mathcal{A}(z^{1:L}), \Sigma_y), \quad (3.7)$$

such that  $\mathcal{A}(z^{1:L}) = (\mathcal{M}^1(D_\psi(z^1)) \cdots \mathcal{M}^L(D_\psi(z^L)))^\top$  and  $\Sigma_y$  is the covariance of  $\eta^{1:L}$ . With this formulation, off-the-shelf posterior sampling algorithms [28] can be used for generating atmospheric trajectories conditionally on observational data. In this work, we adapt moment matching posterior sampling (MMPS), originally proposed by Rozet et al. [27] for linear observation operators. In our case, since  $\mathcal{A}$  is non-linear, we use its Jacobian  $A$  in the estimate of the covariance, yielding the approximation of the perturbed likelihood

$$p(y^{1:L} | z_t^{1:L}) \approx \mathcal{N}(y^{1:L} | \mathcal{A}(\mathbb{E}[z^{1:L} | z_t^{1:L}]), \Sigma_y + A \mathbb{V}[z^{1:L} | z_t^{1:L}] A^\top). \quad (3.8)$$

After inference in the latent space, the generated trajectories  $z^{1:L}$  are decoded back to the atmospheric space via the decoder  $\hat{x}^i = D_\psi(z^i)$ .

### 3.3 EXPERIMENTS

We train and evaluate Appa on the ERA5 reanalysis dataset [29] following standard chronological splitting: 1993–2019 for training, 2020 for validation, and 2021–2023 for

testing. Below, we evaluate the latent representation quality, assimilation performance across reanalysis, filtering and forecasting tasks, and compare against existing methods.

**Latent representation** Despite the  $530\times$  compression factor, standardized reconstruction root-mean-squared errors (RMSE) are mostly below 0.1, with slightly higher values for humidity and winds, and lower ones for surface and low-altitude fields, as shown in Figure 3.1. Reconstructed power spectra match the ground truth closely, except at scales below 100 km where atmospheric energy is minimal. Compared to prior neural compression methods [30], our autoencoder achieves comparable performance.

**Assimilation** We evaluate Appa across four scenarios: reanalysis, filtering, observational forecasting, and full-state forecasting. For the first three tasks, we assimilate both synthetic ground-station observations of all 6 surface fields and simulated scans of the 5 atmospheric fields across 13 pressure levels. The ground station network consists of 11 000 real-world measurement locations [31] covering roughly 1 % of grid points. Ground stations are sparse and globally distributed, while satellite orbital paths provide dense spatial coverage with restricted temporal and spatial reach. Observations are modeled as Gaussian distributions centered on the ERA5 ground truth, with noise levels of 1 % for ground stations and 10 % for satellite measurements.

Figure 3.2 summarizes Appa’s performance and further qualitative results can be found in Appendix 3.D.4. For reanalysis and filtering, conditioning on longer assimilation windows improves both skill and CRPS but gains saturate beyond 24 hours. Forecasting’s skill decays gradually with lead time but remains significantly stronger than the persistence baseline. Observational forecasts, conditioned on the last 12 hours of a day-long assimilation, start at skill and CRPS levels comparable to the reanalysis plateau. Full-state forecasts, initialized from two complete states, start lower but eventually converge to similar performance over time. As expected, the initial skill for

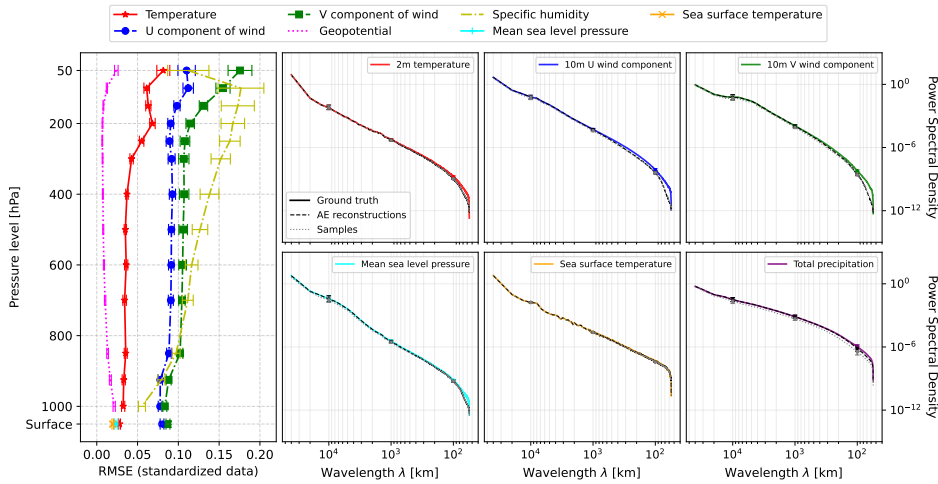


Figure 3.1. (Left) Standardized autoencoder reconstruction RMSE. Lower-frequency fields (temperature, geopotential) are reconstructed more accurately than volatile fields (humidity, winds). Near-surface fields benefit from altitude-weighting. (Right) Power spectral density comparison of ground truth, autoencoder reconstructions, and samples generated from Appa’s prior. Median and percentile ranges show close alignment across scales, with deviations below 100 km (3 to 4 grid cells).

full-state forecasts is close to autoencoder reconstruction error levels. Overall, these results demonstrate that Appa successfully handles all assimilation and forecasting tasks within a unified probabilistic framework.

For short lead times, our forecasts reach skill levels comparable to IFS [32] while performing better than GraphDOP [33]. Notably, the forecast error growth lies between the two baselines: slightly steeper than IFS but below GraphDOP and closely tracking its slope. This suggests that compression does not introduce strong dynamical artifacts that impede the learning of the dynamics, even under different temporal setups (1-hour for Appa vs. 12-hour resolution for IFS), which corroborates the findings of Rozet et al. [17]. While these results remain preliminary, they provide promising evidence that Appa can capture atmospheric dynamics at a level in between purely physical and purely data-driven approaches.

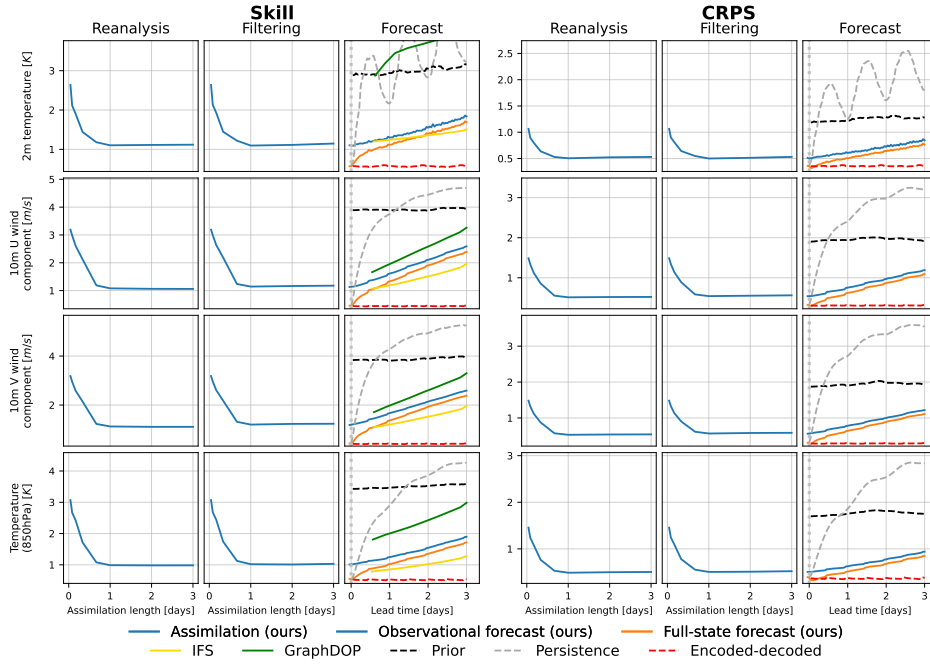


Figure 3.2. Skill and continuous ranked probability score (CRPS) for representative fields in January 2023. The detailed experimental setup can be found in Appendix 3.B.3. Reanalysis scores are averaged over assimilation windows  $x^{1:L}$ , while filtering reports the last reanalyzed state. Both improve with longer assimilation windows, but eventually stagnate. Forecasts gradually lose skill over lead time but remain superior to persistence and unconditional baselines. The initial skill level of the full-state forecast matches the compression error level. IFS [32] and GraphDOP [33] are shown for reference.

### 3.4 DISCUSSION

**Summary** We introduce Appa, a latent score-based data assimilation framework that produces global atmospheric trajectories by operating in a compressed latent space. Appa can be conditioned on various types of observations without retraining, providing access to the full posterior distribution of consistent trajectories. Our results show that Appa flexibly handles reanalysis, filtering, and forecasting within a single framework, producing competitive performance across scenarios without task-specific training or architectural modifications.

**Limitations and future work** While Appa demonstrates strong assimilation and forecasting capabilities, it remains a proof of concept and further improvements are needed to make it operational. First, we should consider moving from simplified synthetic observations to realistic measurements, such as satellite radiances. Improving physical consistency is also critical, as compression inevitably degrades fine-scale information. Strategies such as localized assimilation or refined conditioning mechanisms may help. In terms of statistical assessment, the calibration of posterior distributions deserves further validation. Indeed, the approximations present in our method, notably while estimating the prior and likelihood scores, introduce errors which are complex to quantify. The computational efficiency of observational tasks remains a challenge as well, since conditioning requires repeated decoding steps; projecting observations directly into latent space could mitigate this bottleneck. Finally, our comparison to other models is still preliminary. Baselines for assimilation remain scarce, and fair evaluation against IFS [32], GraphDOP [33] and other models [18, 19, 34] would help position Appa within the spectrum of global atmospheric models.

## ACKNOWLEDGMENTS

G r me Andry, Fran ois Rozet, Sacha Lewin, and Elise Faulx are research fellows of the F.R.S.-FNRS (Belgium) and acknowledge its financial support. Omer Rochman acknowledges the financial support of the Walloon Region under grant no. 2.0102.35 (ARIAC by Digital Wallonia 4.AI). Victor Mangeleer is a research fellow part of the Multiple THreats on Ocean health (MiTHO) project and acknowledges funding from the European Space Agency (ESA).

The present research benefited from computational resources made available on Lucia, the Tier-1 supercomputer of the Walloon Region, infrastructure funded by the Walloon Region under the grant no. 1910247. The computational resources have been provided by the Consortium des  quipements de Calcul Intensif (C CI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under the grant no. 2.5020.11 and by the Walloon Region.

## REFERENCES

- [1] G r me Andry et al. “Appa: Bending Weather Dynamics with Latent Diffusion Models for Global Data Assimilation”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025.
- [2] A. C. Lorenc. “Analysis methods for numerical weather prediction”. In *Quarterly Journal of the Royal Meteorological Society* 112.474 (1986).
- [3] Fran ois-Xavier Le Dimet and Olivier Talagrand. “Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects”. In *Tellus A* 38A.2 (1986).
- [4] Yannick Tr molet. “Accounting for an imperfect model in 4D-Var”. In *Quarterly Journal of the Royal Meteorological Society* 132.621 (2006).
- [5] Yannick Tr molet. “Model-error estimation in 4D-Var”. In *Quarterly Journal of the Royal Meteorological Society* 133.626 (2007).
- [6] Mike Fisher et al. “Weak-constraint and long window 4DVAR”. Tech. rep. ECMWF, 2011.
- [7] Brian R. Hunt, Eric J. Kostelich, and Istvan Szunyogh. “Efficient data assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter”. In *Physica D: Nonlinear Phenomena. Data Assimilation* 230.1 (2007).
- [8] Alberto Carrassi et al. “Data assimilation in the geosciences: An overview of methods, issues, and perspectives”. In *WIREs Climate Change* 9 (2018).
- [9] Sibo Cheng et al. “Machine Learning With Data Assimilation and Uncertainty Quantification for Dynamical Systems: A Review”. In *IEEE/CAA Journal of Automatica Sinica* 10.6 (2023).
- [10] Langwen Huang et al. “DiffDA: a Diffusion model for weather-scale Data Assimilation”. In *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024.
- [11] Xiaoze Xu et al. “FuXi-DA: a generalized deep learning data assimilation framework for assimilating satellite observations”. In *npj Climate and Atmospheric Science* 8.1 (2025).
- [12] Ronan Fablet et al. “Joint Interpolation and Representation Learning for Irregularly Sampled Satellite-Derived Geophysical Fields”. In *Frontiers in Applied Mathematics and Statistics* 7 (2021).
- [13] Marcin Andrychowicz et al. “Deep Learning for Day Forecasts from Sparse Observations”. 2023.
- [14] Fran ois Rozet and Gilles Louppe. “Score-based Data Assimilation”. In *Advances in Neural Information Processing Systems*. Vol. 36. 2023.
- [15] Fran ois Rozet and Gilles Louppe. “Score-based Data Assimilation for a Two-Layer Quasi-Geostrophic Model”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2023.
- [16] Jonathan Schmidt et al. “A Generative Framework for Probabilistic, Spatiotemporally Coherent Downscaling of Climate Simulation”. In *npj Climate and Atmospheric Science* 8.1 (2025).
- [17] Fran ois Rozet et al. “Lost in Latent Space: An Empirical Study of Latent Diffusion Models for Physics Emulation”. In *Advances in Neural Information Processing Systems*. Vol. 38. 2025.

- [18] Remi Lam et al. “Learning skillful medium-range global weather forecasting”. In *Science* 382.6677 (2023).
- [19] Ilan Price et al. “Probabilistic weather forecasting with machine learning”. In *Nature* 637.8044 (2025).
- [20] William Peebles and Saining Xie. “Scalable Diffusion Models with Transformers”. In *IEEE/CVF International Conference on Computer Vision*. 2023.
- [21] Tero Karras et al. “Elucidating the Design Space of Diffusion-Based Generative Models”. In *Advances in Neural Information Processing Systems*. Vol. 35. 2022.
- [22] M. C. K. Tweedie. “Functions of a statistical variate with given means, with special reference to Laplacian distributions”. In *Mathematical Proceedings of the Cambridge Philosophical Society* (1947).
- [23] Bradley Efron. “Tweedie’s Formula and Selection Bias”. In *Journal of the American Statistical Association* (2011).
- [24] Kwanyoung Kim and Jong Chul Ye. “Noise2Score: Tweedie’s Approach to Self-Supervised Image Denoising without Clean Images”. In *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021.
- [25] Yang Song et al. “Score-Based Generative Modeling through Stochastic Differential Equations”. In *International Conference on Learning Representations*. 2021.
- [26] Hyungjin Chung et al. “Diffusion Posterior Sampling for General Noisy Inverse Problems”. In *International Conference on Learning Representations*. 2023.
- [27] François Rozet et al. “Learning Diffusion Priors from Observations by Expectation Maximization”. In *Advances in Neural Information Processing Systems*. Vol. 37. 2024.
- [28] Giannis Daras et al. “A Survey on Diffusion Models for Inverse Problems”. 2024.
- [29] Hans Hersbach et al. “The ERA5 global reanalysis”. In *Quarterly Journal of the Royal Meteorological Society* 146.730 (2020).
- [30] Piotr Mirowski et al. “Neural Compression of Atmospheric States”. 2024.
- [31] NOAA National Centers of Environmental Information. “Global Surface Summary of the Day - GSOD”. 1999.
- [32] ECMWF. “IFS documentation CY47R1 - part II: Data assimilation”. In *IFS Documentation CY47R1*. ECMWF, 2020.
- [33] Mihai Alexe et al. “GraphDOP: Towards skilful data-driven medium-range weather forecasts learnt and initialised directly from observations”. 2024.
- [34] Ron Raphaeli, Sean Man, and Michael Elad. “SILO: Solving Inverse Problems with Latent Operators”. 2025.
- [35] Stephan Rasp et al. “WeatherBench 2: A Benchmark for the Next Generation of Data-Driven Global Weather Models”. In *Journal of Advances in Modeling Earth Systems* 16.6 (2024).
- [36] Nikhil Vyas et al. “SOAP: Improving and Stabilizing Shampoo using Adam for Language Modeling”. In *International Conference on Learning Representations*. 2025.
- [37] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In *International Conference on Learning Representations*. 2015.
- [38] V. Fortin et al. “Why Should Ensemble Spread Match the RMSE of the Ensemble Mean?” In *Journal of Hydrometeorology* 15.4 (2014).
- [39] Hans Hersbach. “Decomposition of the Continuous Ranked Probability Score for Ensemble Prediction Systems”. In *Weather and Forecasting* 15.5 (2000).



- [40] Tilmann Gneiting and Adrian E Raftery. “Strictly Proper Scoring Rules, Prediction, and Estimation”. In *Journal of the American Statistical Association* 102.477 (2007).

## 3.A DATA

### 3.A.1 ERA5

ERA5 is a global deterministic reanalysis dataset from ECMWF that provides high-resolution ( $0.25^\circ$ ) hourly estimates of atmospheric, land, and oceanic fields from 1959 onward [29]. It assimilates observations into a numerical weather prediction model using 4D-Var data assimilation.

For this work, we use a subset of ERA5 data, defined on a  $0.25^\circ$  equiangular grid with 13 pressure levels: 50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 850, 925, and 1000 hPa. Due to storage limitations, we restrict the temporal coverage of the dataset to the 1993–2023 period, with data split into training (1993–2019), validation (2020), and testing (2021–2023).

Table 3.1 lists the selected fields. Some serve as both input and predicted features, while others provide contextual information (only input). Context fields are not predicted but help define the temporal and dynamic conditions under which predictions are made, improving model performance.

Table 3.1. Input fields.

| Type        | Name                    | Role              |
|-------------|-------------------------|-------------------|
| Atmospheric | Temperature             | Input / Predicted |
| Atmospheric | U-Wind Component        | Input / Predicted |
| Atmospheric | V-Wind Component        | Input / Predicted |
| Atmospheric | Geopotential            | Input / Predicted |
| Atmospheric | Specific Humidity       | Input / Predicted |
| Single      | 2m Temperature          | Input / Predicted |
| Single      | 10m U-Wind Component    | Input / Predicted |
| Single      | 10m V-Wind Component    | Input / Predicted |
| Single      | Mean Sea Level Pressure | Input / Predicted |
| Single      | Sea Surface Temperature | Input / Predicted |
| Single      | Total Precipitation     | Input / Predicted |
| Clock       | Local time of day       | Input (diffusion) |
| Clock       | Elapsed year progress   | Input (diffusion) |

### 3.A.2 DATA PRE-PROCESSING

**Standardization** Although the dynamics across the atmospheric column are correlated, each pressure level exhibits distinct statistical behavior. Thus, we computed the mean and standard deviation separately for each field and at each pressure level, on the whole training dataset. We used these statistics to standardize our entire dataset and to rescale the output of Appa.

**Handling missing values** Since sea surface temperature is undefined over land (NaN values), we replace these with zeros as a neutral placeholder after standardization.

**Data availability** ERA5 data was downloaded from the WeatherBench2 [\[35\]](#) platform, where Google has made them publicly available via Google Cloud Storage.

### 3.B TECHNICAL DETAILS

This section provides further technical details for training and inference. Our code will be made available with full reproducibility steps for both training and evaluation.

#### 3.B.1 ARCHITECTURES

We adapted architectures from Rozet et al. [17] for both autoencoder and latent diffusion model. The encoder and decoder are fully convolutional neural networks and the diffusion model is adapted from a diffusion transformer (DiT) [20].

**Autoencoder** The autoencoder compresses atmospheric states from the high-dimensional N320 grid ( $721 \times 1440$  pixels with 71 channels) into a compact latent space ( $23 \times 47$  pixels with 128 channels) via progressive downsampling and channel expansion in a fully convolutional architecture. To accommodate any spatial compression factor, the input is padded to the nearest compatible grid size. We apply periodic padding along longitude to respect global wrap-around, and constant zero padding along latitude to handle polar boundaries. The encoder-decoder pair is trained with a latitude-level-weighted mean squared error loss, following Lam et al. [18].

**Latent denoiser** The denoiser is a DiT that operates on 24 consecutive latent states. We first patch the latent sequence by a factor of 2 along the temporal axis, then flatten the spatial dimensions, yielding  $23 \times 47 \times 24/2 = 12\,972$  tokens, each with 256 channels, which are passed to the DiT.

Table 3.2. Autoencoder training configuration

| Parameter         | Value                                                                    |
|-------------------|--------------------------------------------------------------------------|
| Loss function     | Latitude- and level-weighted mean squared error                          |
| Latent noise      | $\sigma_z = 0.01$ for regularization                                     |
| Optimizer         | SOAP [36] with initial learning rate $3 \times 10^{-5}$ and linear decay |
| Batch size        | 64 samples per step                                                      |
| Training duration | 95000 update steps (approximately 2 days)                                |
| Hardware          | 64 A100 (40GB) GPUs                                                      |

Table 3.3. Denoiser training configuration

| Parameter         | Value                                                  |
|-------------------|--------------------------------------------------------|
| Loss              | Denoising score matching with rectified noise schedule |
| Noise range       | $\sigma_{\min} = 10^{-3}$ , $\sigma_{\max} = 10^3$     |
| Optimizer         | Adam [37] with initial learning rate $10^{-4}$         |
| Batch size        | 256 samples per step                                   |
| Training duration | 125000 update steps (approximately 5 days)             |
| Hardware          | 64 A100 (40GB) GPUs                                    |

## 3.B.2 GENERATING TRAJECTORIES

**Algorithm 2** Training  $d_\phi(z_t^{i:i+W})$ 


---

```

1 for  $n = 1$  to  $N$  do
2    $x^{1:L} \sim p(x^{1:L})$ 
3    $i \sim \mathcal{U}(\{1, \dots, L - W\})$ 
4    $t \sim \mathcal{U}(0, 1), \varepsilon \sim \mathcal{N}(0, I)$ 
5   for  $j = i$  to  $i + W$  do
6      $z_j \leftarrow E_\psi(x_j)$ 
7      $z_t^{i:i+W} \leftarrow z^{i:i+W} + \sigma_t \varepsilon$ 
8      $\ell \leftarrow \|d_\phi(z_t^{i:i+W}) - z^{i:i+W}\|_2^2$ 
9      $\phi \leftarrow \text{SGD}(\phi, \nabla_\phi \ell)$ 

```

---

**Algorithm 3** Composing  $d_\phi(z_t^{i:i+W})$ 


---

```

1 function  $d_\phi(z_t^{1:L})$ 
2    $a \leftarrow (W - \Delta)/2$ 
3    $b \leftarrow a + \Delta$ 
4    $E_{1:a} \leftarrow d_\phi(z_t^{1:1+W})[ : a]$ 
5   for  $n = 0$  to  $(L - W)/\Delta + 1$  do
6      $i \leftarrow 1 + n\Delta$ 
7      $E_{i+a:i+b} \leftarrow d_\phi(z_t^{i:i+W})[a : b]$ 
8    $E_{L-W+b:L} \leftarrow d_\phi(z_t^{L-W:L})[b : ]$ 
9   return  $E_{1:L}$ 

```

---

## 3.B.3 ASSIMILATION TASKS

**Forecasting** Appa is trained to generate state windows of a given size; we use 24 hours. We first split the total window in two, the first part being the condition, and the second the part to be generated. Then, we use either the last states of a fully assimilated window (observational forecasting) or full encoded latent states (full-state forecasting). At each autoregressive step, we use a sliding window to move  $n$  steps forward after these steps were predicted. This mechanism offers a balance between conditioning window size and generation speed, as the former can be extended to provide more context but more limited speed (more generations required) or less context but faster total generation.

**Evaluation setup** In Figure 3.2, we report IFS [32] and GraphDOP [33] skills using data from Alexe et al. [33] as baselines. However, a direct and fair comparison is difficult due to a lack of experimental setup details. Alexe et al. [33] mention that skills are computed over January 2023 forecasts for 6 different fields. To match the time period, we report the performance of our method over a 10-member ensemble of 3-day forecasts. We selected the first 8 days of January 2023 at midnight as starting timestamps. This makes a total of 8 dates with 10 members for reported metrics.

### 3.C EVALUATION METRICS

We follow conventional metrics computation for assimilation and forecasting performance. For a fair comparison with the literature, evaluation is performed using WeatherBench2 [35]. For assimilation, we average performance over the time steps.

**Skill** Skill is computed as the root mean square error of the posterior mean of an ensemble compared to the ground-truth trajectory. For  $K$  ensembles each consisting of  $M$  predicted states  $\hat{x}$  of resolution  $H \times W$ , and ground truth  $x$ , the skill of a single time step is computed as

$$\text{Skill} = \sqrt{\frac{1}{KHW} \sum_{k=1}^K \sum_{i=1}^H \sum_{j=1}^W \left( x_{i,j}^k - \frac{1}{M} \sum_{m=1}^M \hat{x}_{i,j}^{k,m} \right)^2}. \quad (3.9)$$

**Spread** Ensemble spread is computed as the square root of the ensemble variance [38]

$$\text{Spread} = \sqrt{\frac{1}{KHW} \sum_{k=1}^K \sum_{i=1}^H \sum_{j=1}^W \frac{1}{M-1} \sum_{m=1}^M \left( \hat{x}_{i,j}^{k,m} - \frac{1}{M} \sum_{n=1}^M \hat{x}_{i,j}^{k,n} \right)^2}. \quad (3.10)$$

**Spread-skill ratio** A well-calibrated forecast should have a (corrected for ensemble size) spread-skill ratio of 1, which is a necessary but not sufficient condition. Ratios below one indicate overconfident estimations. The corrected ratio is defined as

$$\text{Ratio} = \sqrt{\frac{M+1}{M} \frac{\text{Spread}}{\text{Skill}}}. \quad (3.11)$$

**Continuous ranked probability score (CRPS)** The CRPS [39, 40] is defined as

$$\text{CRPS} = \frac{1}{K} \sum_{k=1}^K \left( \frac{1}{M} \sum_{m=1}^M \|\hat{x}^{k,m} - x^k\|_{L_1} - \frac{1}{2M(M-1)} \sum_{m=1}^M \sum_{n=1}^M \|\hat{x}^{k,m} - \hat{x}^{k,n}\|_{L_1} \right). \quad (3.12)$$

The first term penalizes the average divergence from the ground truth while the second term encourages spread. Therefore, the CRPS is lowest when the distribution of the ensemble matches the ground-truth distribution. Note the  $L_1$  norm used, which means that in the deterministic case, this reduces to the MAE.

## 3.D ADDITIONAL RESULTS

## 3.D.1 POWER SPECTRAL DENSITY

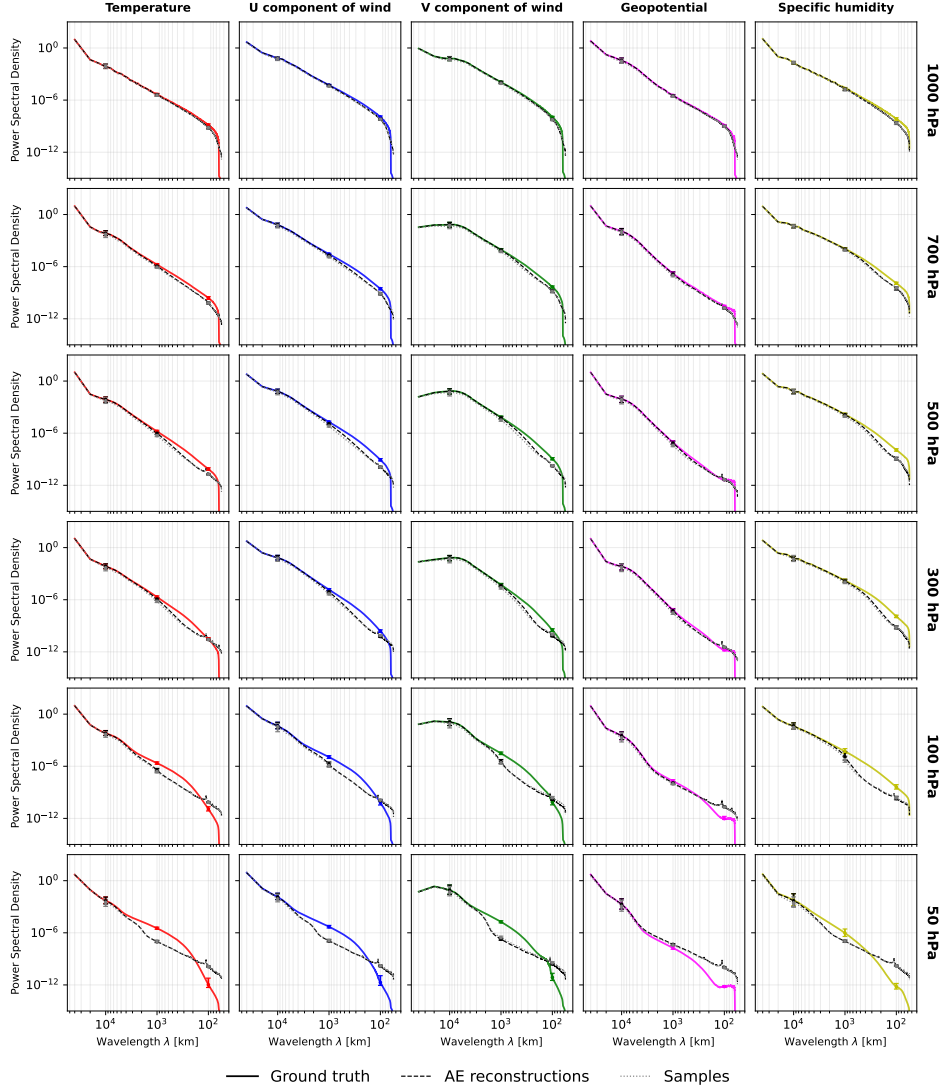


Figure 3.3. Power spectral density across wavelengths for atmospheric fields at selected pressure levels. Lines show median values and error bars indicate the 5th to 95th percentiles. The close alignment between the curves demonstrates that both the autoencoder and the diffusion model preserve the energy distribution across most spatial scales. Deviations begin to appear at wavelengths around 1000 km, which corresponds to roughly 40 grid cells at our  $0.25^\circ$  resolution at the equator. These differences become more pronounced at smaller scales, suggesting that while large-scale atmospheric patterns are well-preserved, features spanning fewer than 40 grid cells show some energy loss in the compression and generation processes. Deviations become more pronounced at lower pressure levels, as the model prioritizes surface and low-altitude fields.

## 3.D.2 PHYSICAL CONSISTENCY

To further evaluate physical consistency, we examine whether our model preserves important physical relationships between fields. First, we analyze the consistency between two different estimators of altitude at given pressure levels. Using the geopotential  $\Phi$ , altitude can be derived as

$$H = \frac{\Phi R_e}{g_0 R_e - \Phi}, \quad (3.13)$$

where  $R_e$  is Earth’s radius and  $g_0$  is Earth’s gravitational acceleration at the surface. Alternatively, the equation below (which relies on the ideal gas law and hydrostatic equation) relates altitude to pressure and temperature as

$$\log \frac{p_0}{p_H} = \frac{M g_0}{R} \int_0^H \frac{1}{T_h} \partial h, \quad (3.14)$$

where  $R$  is the universal gas constant,  $M$  is an approximation of the atmosphere’s molar mass,  $p_h, T_h$  are pressure and temperature at height  $h$ , and  $p_0, T_0$  are the theoretical pressure and temperature at sea level. This integral can be approximated to extract  $H$  using several assumptions about the temperature profile. When comparing these two estimators, Figure 3.4 shows that our generated samples maintain the same systematic differences  $\Delta H$  as seen in ground-truth data. This remarkable consistency indicates that our model successfully preserves this physical relationship between temperature, pressure, and geopotential, allowing altitude to be estimated through two independent methods with nearly identical accuracy to the original ERA5 data.

Second, we examine the geostrophic balance

$$\frac{\partial \Phi}{\partial x} = \frac{4\pi\Omega R_e}{N_x} \sin \phi \cos \phi u_g \quad (3.15)$$

$$\frac{\partial \Phi}{\partial y} = -\frac{2\pi\Omega R_e}{N_y} \sin \phi v_g \quad (3.16)$$

which is the theoretical equilibrium between pressure gradient forces and Coriolis forces that governs large-scale atmospheric motion. In the above system,  $\phi$  is the latitude,  $N_x$  and  $N_y$  are the number of pixels along longitude and latitude,  $\Omega$  is the magnitude of the Earth’s angular velocity, and  $u_g$  and  $v_g$  denote the geostrophic components of the wind. In this balance, in the absence of vertical motion, friction, and isobaric curvature, wind direction should be perpendicular to geopotential gradients, with wind speed proportional to gradient magnitude. This relationship can be expressed by comparing two quantities: (1) the angle  $\theta$  between wind and geopotential gradients, which should approach  $90^\circ$  in geostrophic conditions, and (2) the correlation between wind speed magnitude and geopotential gradient magnitude, which should approach 1 in perfect geostrophic balance. Figure 3.4 shows that our generated samples accurately reproduce both aspects of this relationship. At 500 hPa, the approximate level of non-divergence with minimal surface friction effects, both ERA5 data and our generated samples show angles concentrated around  $90^\circ$ . Near the surface at 1000 hPa, where additional forces become significant, both datasets show a systematic deviation in angle. Similarly, the correlation between wind speed and geopotential gradient magnitudes in our samples closely matches the patterns observed in ERA5 data, exhibiting imperfect correlation only at lower pressure levels (explained by the level-weighted training) and following the same decreasing trend as pressure increases toward the surface, where ageostrophic components become more prominent.

These results demonstrate that our latent diffusion model not only preserves the statistical properties of atmospheric fields but also maintains important physical relationships between fields, producing trajectories that are physically consistent and realistic. While



these analyses confirm strong spatial consistency and physical fidelity, future work should extend our evaluation to more thoroughly quantify the temporal consistency of generated trajectories.

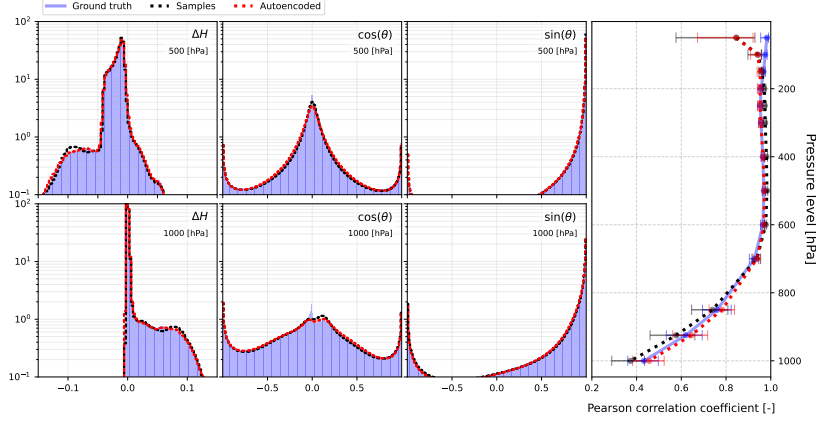


Figure 3.4. Physical consistency analysis of generated atmospheric states. **(Top row)** Analysis of altitude consistency at 500 hPa showing the difference  $\Delta H$  between two independent altitude estimators, and geostrophic balance assessment through the cosine and sine of the angle  $\theta$  between wind direction and geopotential gradients, demonstrating angles concentrated around  $90^\circ$ . **(Bottom row)** Same metrics at 1000 hPa demonstrating the presence of a significant ageostrophic component near the surface. **(Right)** Correlation coefficient between wind magnitude and geopotential gradient magnitude across pressure levels, showing strong correlation at upper levels (near 1) with a consistent decrease toward the surface in both ERA5 data (blue) and generated samples (black dots), confirming Appa’s ability to capture complex physical relationships.

## 3.D.3 QUANTITATIVE EVALUATION

## COMPRESSION ERROR

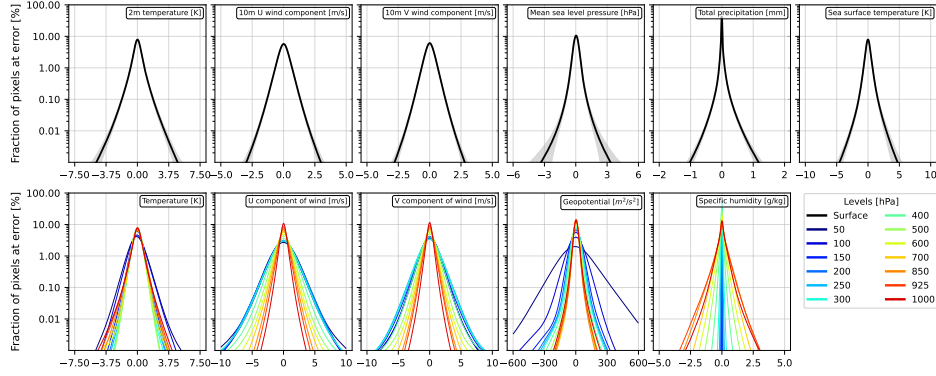


Figure 3.5. Signed reconstruction errors for surface fields (top) across all grid points and atmospheric fields (bottom) across all pressure levels. The shaded gray area for surface fields corresponds to error spread. In both cases, the concentrated distributions centered around zero demonstrate unbiased and precise predictions. Given  $721 \times 1440 = 1\,038\,240$  grid points, a 0.01 % fraction on the y-axis corresponds to approximately 100 grid points, indicating that large errors are rare.

## ADDITIONAL FIELDS

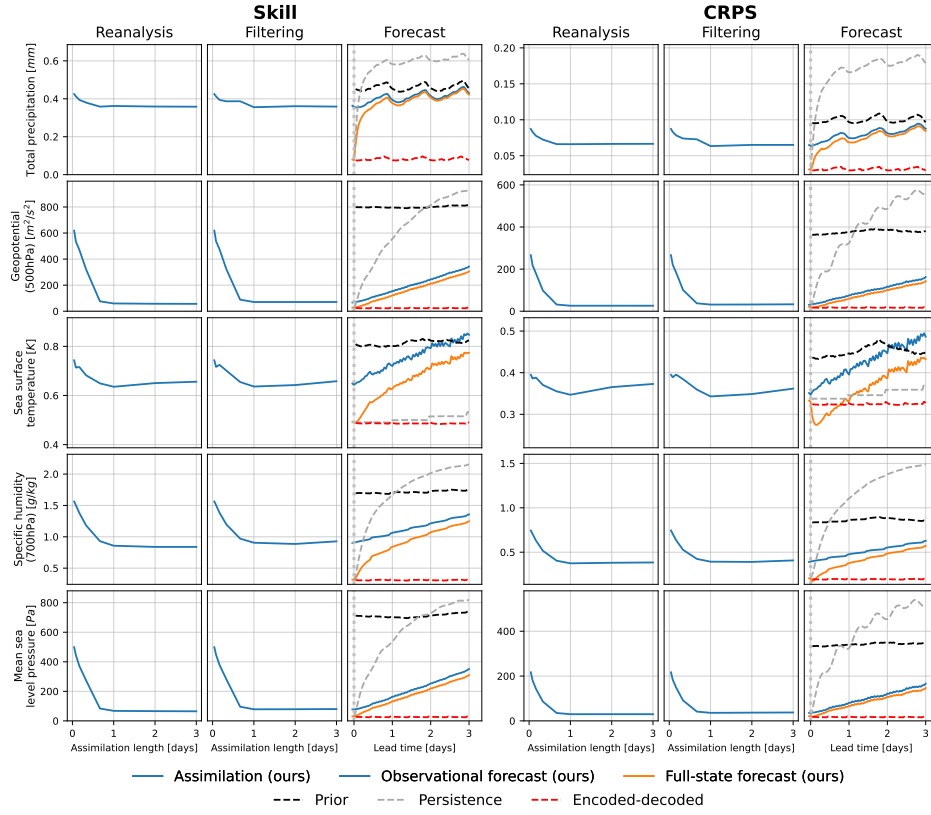


Figure 3.6. Quantitative evaluation of reanalysis, filtering, and forecasting for additional representative fields. (Left) Skill score and (Right) continuous ranked probability score (CRPS). Reanalysis scores are averaged over assimilation windows, while filtering reports the last reanalyzed state. Both improve with longer windows. Forecasts gradually lose skill over lead time but remain above persistence and unconditional baselines.

## SPREAD AND SPREAD-SKILL RATIOS

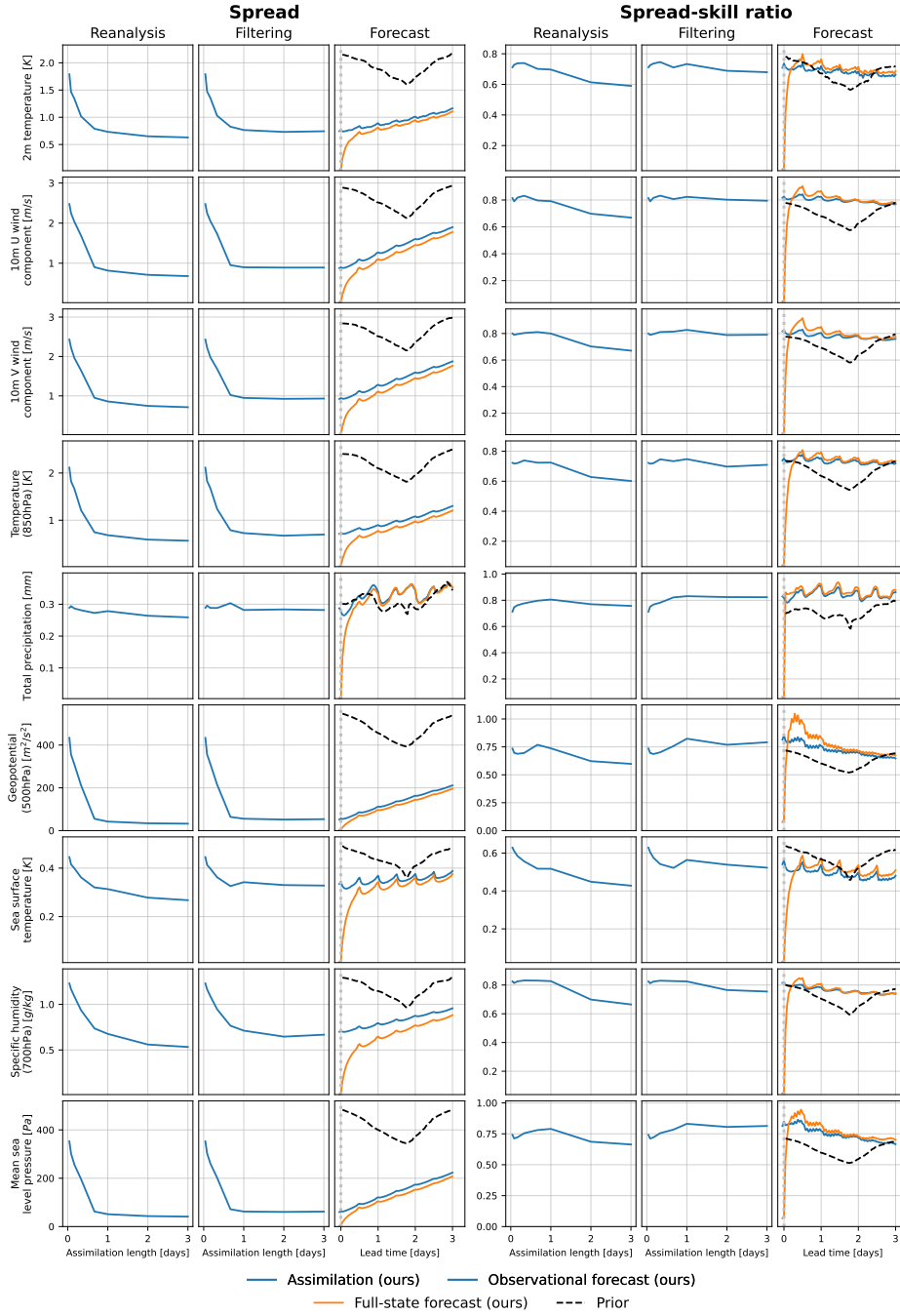


Figure 3.7. Quantitative evaluation of reanalysis, filtering, and forecasting. (Left) Spread and (Right) spread-skill ratio for representative fields. Reanalysis scores are averaged over assimilation windows, while filtering reports the last reanalyzed state. Ensemble spread decreases with longer windows, while ratios remain fairly unchanged. A ratio below one indicates overconfidence.

## 3.D.4 QUALITATIVE SNAPSHOTS

In Figures 3.8 to 3.13, we display decoded sampled trajectories generated through reanalysis over a window of three days as well as through observational forecasting initialized from reanalyzed states, for six representative fields. The second row of each gallery shows the observed pixels, if the state was observed. Each mask is displayed as the corresponding field was observed during inference, i.e., as ground stations for surface fields and satellite scans for atmospheric fields.

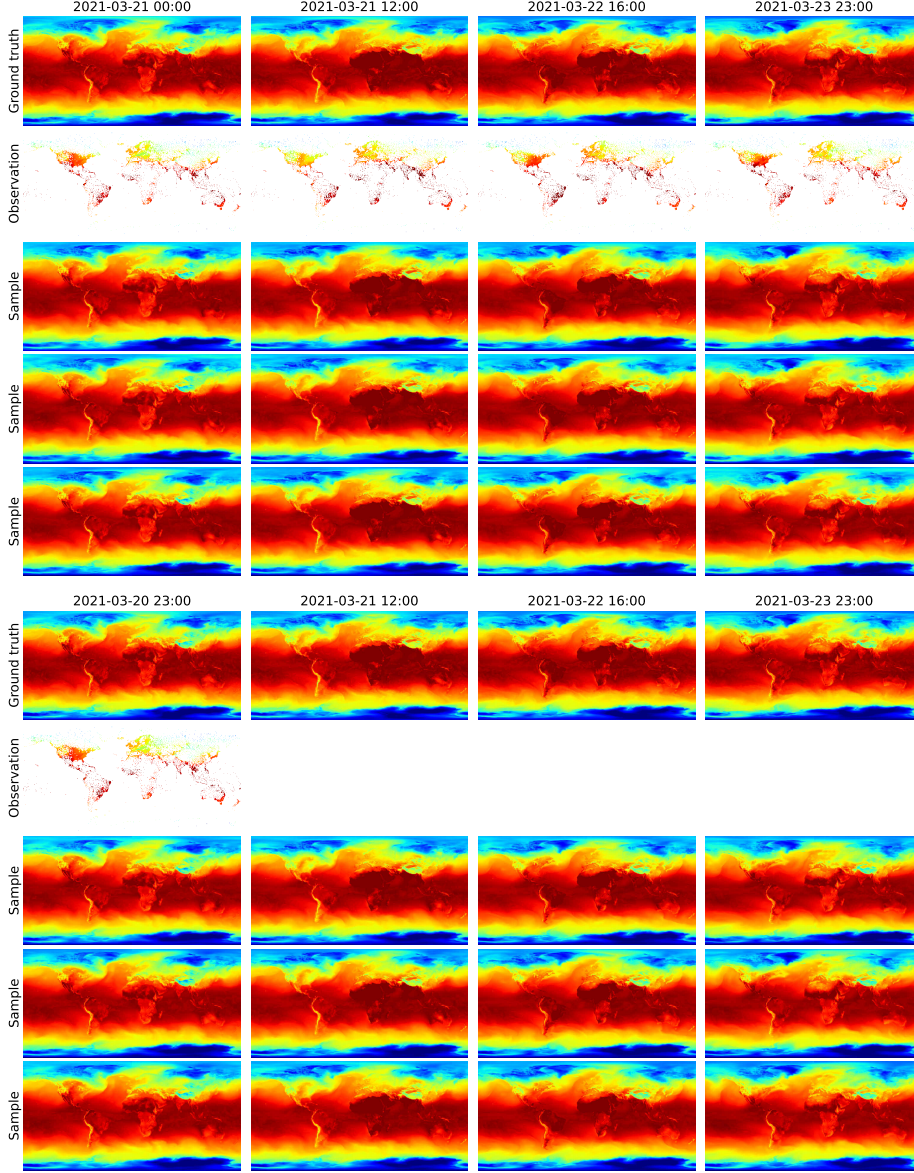


Figure 3.8. Reconstructed sampled trajectories for surface temperature assimilation. (Top) Reanalysis over a window of 72 hours. (Bottom) Observational forecasting over 3 days initialized with the last 12 states of an assimilation over 24 hours.



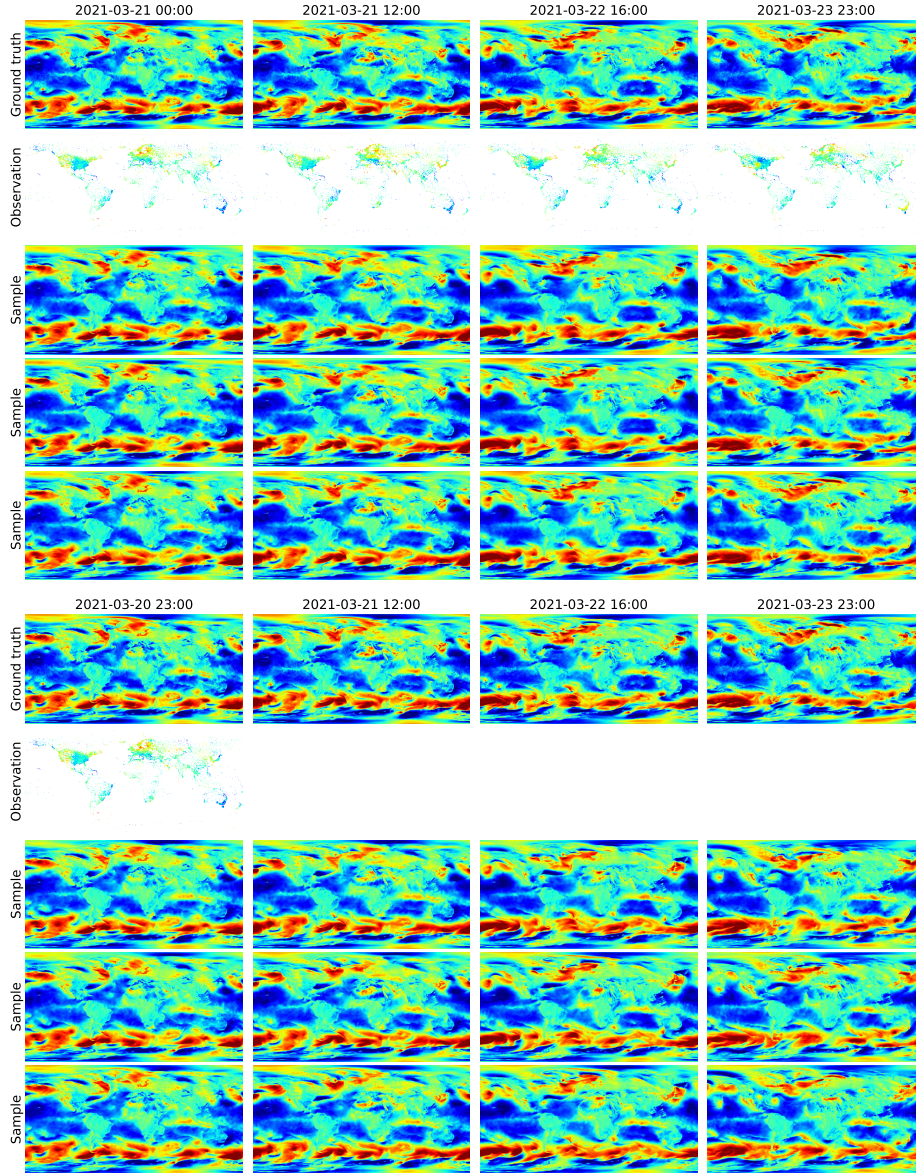


Figure 3.9. Reconstructed sampled trajectories for surface eastward wind speed assimilation. (Top) Reanalysis over a window of 72 hours. (Bottom) Observational forecasting over 3 days initialized with the last 12 states of an assimilation over 24 hours.

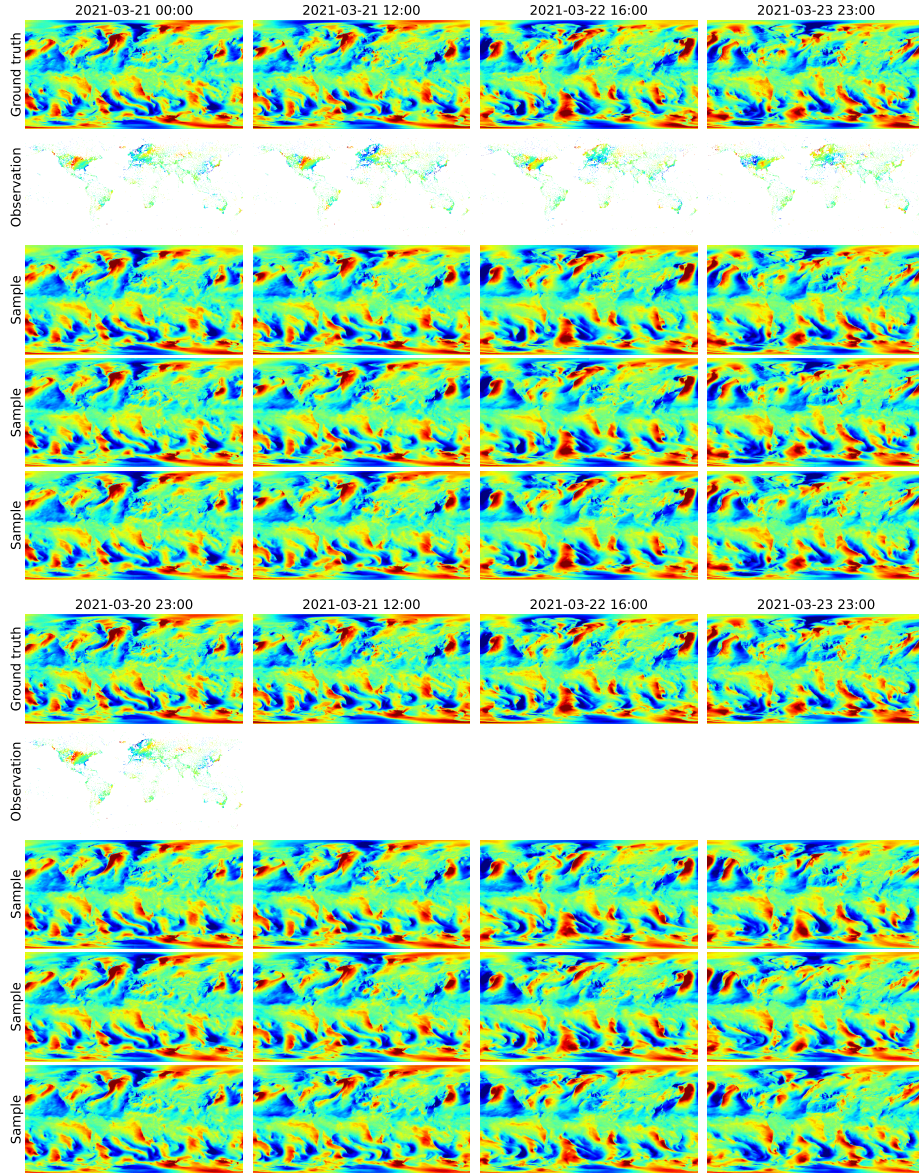


Figure 3.10. Reconstructed sampled trajectories for surface northward wind speed assimilation. (Top) Reanalysis over a window of 72 hours. (Bottom) Observational forecasting over 3 days initialized with the last 12 states of an assimilation over 24 hours.



### 3.D ADDITIONAL RESULTS

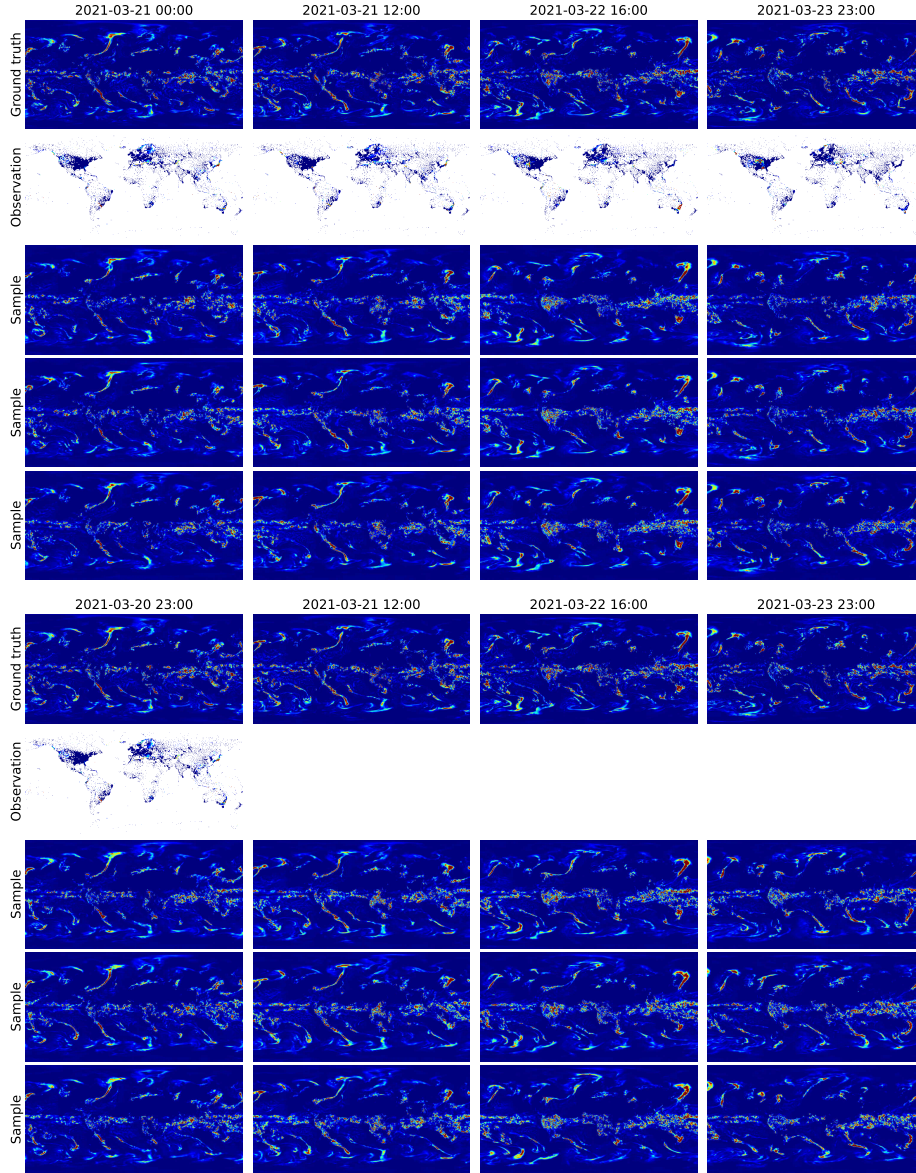


Figure 3.11. Reconstructed sampled trajectories for total precipitation assimilation. (Top) Reanalysis over a window of 72 hours. (Bottom) Observational forecasting over 3 days initialized with the last 12 states of an assimilation over 24 hours.



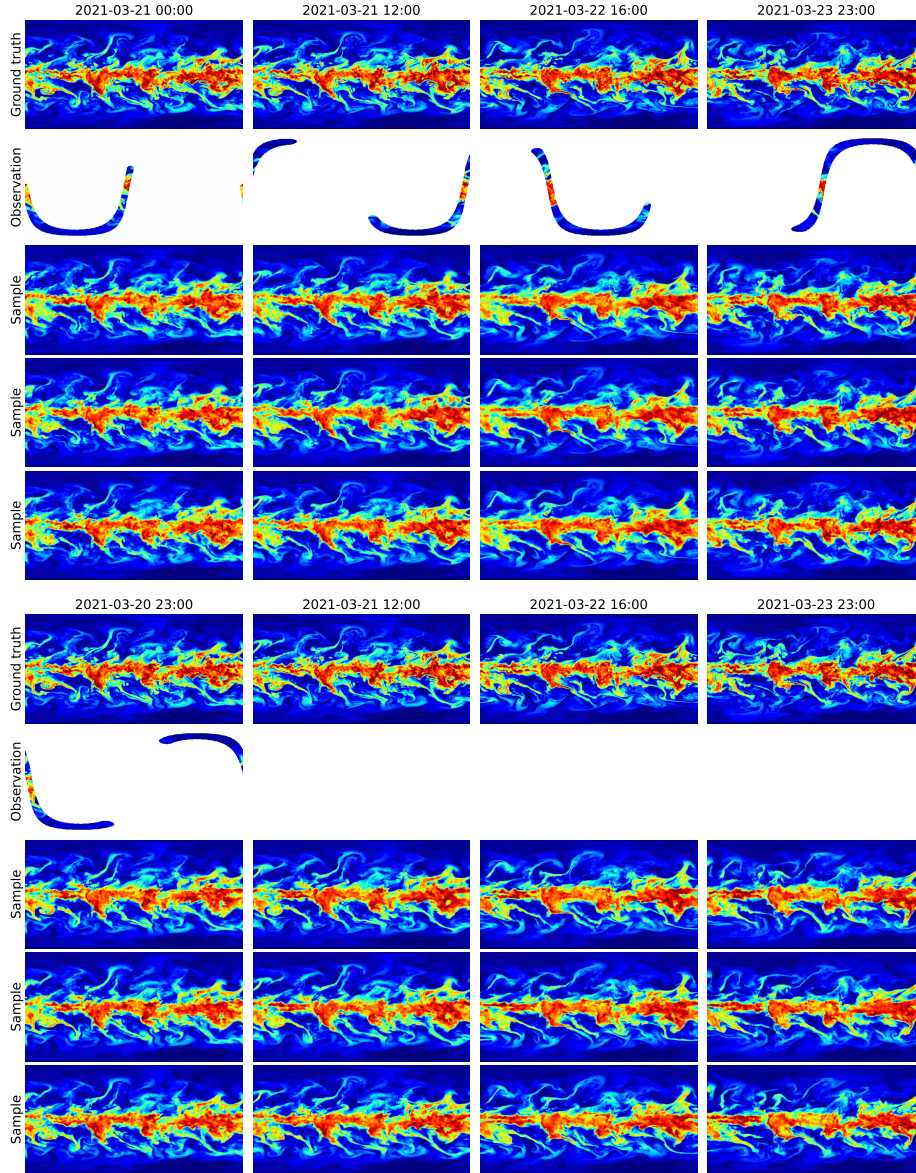


Figure 3.12. Reconstructed sampled trajectories for specific humidity assimilation at 700 hPa. (Top) Reanalysis over a window of 72 hours. (Bottom) Observational forecasting over 3 days initialized with the last 12 states of an assimilation over 24 hours.

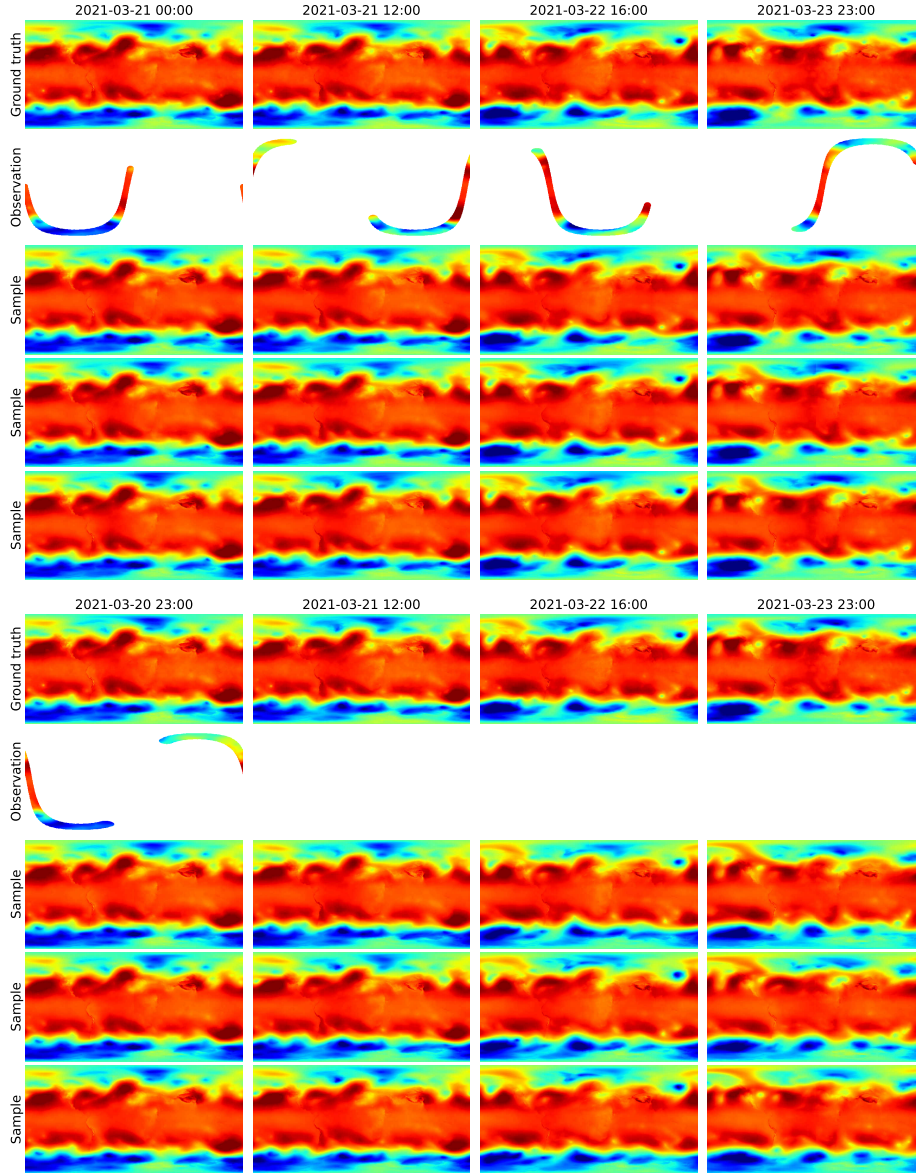


Figure 3.13. Reconstructed sampled trajectories for geopotential assimilation at 850 hPa. (Top) Reanalysis over a window of 72 hours. (Bottom) Observational forecasting over 3 days initialized with the last 12 states of an assimilation over 24 hours.

# 4 ENFORCING GOVERNING EQUATION CONSTRAINTS IN NEURAL PDE SOLVERS VIA TRAINING-FREE PROJECTIONS

*Nada se edifica sobre la piedra, todo sobre la arena, pero nuestro deber es edificar como si fuera piedra la arena.*

— Borges, *Fragmentos de un evangelio apócrifo*

## ADDENDUM

This chapter is based on the paper:

Omer Rochman and Gilles Louppe. “Enforcing governing equation constraints in neural PDE solvers via training-free projections”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025

Omer led this work, devising the method, writing the code, running the experiments, and preparing the manuscript. Gilles supervised the project and participated in the writing.

## ABSTRACT

Neural PDE solvers used for scientific simulation often violate governing equation constraints. While linear constraints can be projected cheaply, many constraints are nonlinear, complicating projection onto the feasible set. Dynamical PDEs are especially difficult because constraints induce long-range dependencies in time. In this work, we evaluate two training-free, post hoc projections of approximate solutions: a nonlinear optimization-based projection, and a local linearization-based projection using Jacobian-vector and vector-Jacobian products. We analyze constraints across representative PDEs and find that both projections substantially reduce violations and improve accuracy over physics-informed baselines.

## 4.1 INTRODUCTION

Neural PDE solvers for scientific simulations often break physical constraints, even when they score well on standard metrics [2–4]. In particle systems this could mean violating mass and momentum conservation [5]; in fluids, incompressibility and adherence to Navier–Stokes [6]; in medical imaging, physiologically plausible ranges for parameters [7].

Depending on the system, physical constraints can be linear or nonlinear, local or global, static or dynamical.

In many problems, physical consistency is best understood as satisfying the constraints implied by the governing PDE, the problem specification, and known physics. Per-timestep conditions, such as divergence-free in incompressible fluids, restrict admissible states but can leave temporal consistency underdetermined, since they hold at each step independently. Conservation laws can help but do not guarantee consistent time evolution. Treating the governing equation itself as a constraint couples space and time and addresses this gap, providing a limit on temporal consistency. In this case, the PDE residual often serves as an informative and, sometimes, the only feasible metric of time consistency, because it dictates how plausible states evolve. Nonetheless, even local near-consistency can result in trajectories that are inconsistent globally.

Existing constraint enforcement methods involve significant tradeoffs. Physics-informed approaches like PINNs [8], PINO [9], and their generative counterparts [10] add penalty terms to the loss function, complicating training and potentially preventing convergence [11]. Architectural enforcement guarantees constraint satisfaction [12] but reduces model expressiveness and requires custom designs for each constraint type. Helper networks [13] introduce additional training overhead and hyperparameter tuning. Constrained sampling methods for generative models [14–17] require backpropagation through sampling paths with repeated constraint evaluations. Moreover, these approaches are rarely validated on complex dynamical constraints like Navier–Stokes, focusing instead on simpler linear constraints such as divergence-free conditions. Finally, PDE determinism can lead to degenerate posteriors with unique solutions, making sampling from such narrow distributions challenging.

**Contributions.** We focus on hard governing equation constraints from nonlinear dynamical PDEs that induce long-range temporal dependencies. We evaluate two training-free, post hoc projection operators of approximate solutions: a nonlinear optimization-based projection and a local linearization-based projection implemented with Jacobian-vector and vector-Jacobian products for scalability. In experiments, both projections substantially reduce violations and improve accuracy over physics-informed training baselines.

*Not everything that can be counted counts, and not everything that counts can be counted.*

— Cameron, *Informal Sociology*, 1963

## 4.2 PROJECTION OPERATORS

We consider a PDE of the general form

$$\mathcal{L}[u] = f, \quad (4.1)$$

over  $\Omega \times (0, T)$ , where  $u$  is the unknown function we seek to find,  $\Omega$  is the spatial domain,  $(0, T)$  is the time interval,  $\mathcal{L}$  is a differential operator that may be nonlinear, and  $f$  is a known source function on  $\Omega \times (0, T)$ . The PDE is subject to initial and boundary conditions  $\mathcal{I}[u] = u_0$  and  $\mathcal{B}[u] = g$ , where  $\mathcal{I}$  and  $\mathcal{B}$  are the initial and boundary condition operators,  $u_0$  specifies the initial state on  $\Omega$ , and  $g$  specifies the boundary values on  $\partial\Omega \times (0, T)$ .

To enforce global consistency, the full solution must be considered at once. Let the discretized state  $u \in \mathbb{R}^N$  be a vector of function evaluations at  $N$  points in space and time,  $u = \{u(x_i, t_i)\}_{i=1}^N$ , where the points  $\{(x_i, t_i) \in \Omega \times (0, T)\}_{i=1}^N$  are chosen to be sufficiently dense to ensure accurate operator discretization. Note that we overload  $u$  to denote both the continuous function and its discretized representation. The discretized PDE problem becomes a system of nonlinear equations  $h(u) = c \in \mathbb{R}^N$  where  $h(u) = [\mathcal{L}_d(u), \mathcal{B}_d(u), \mathcal{I}_d(u)]$  is the concatenation of the discretized operators and  $c = [f_d, g_d, u_{0,d}]$  is the concatenation of the discretized right-hand sides.



Given the output  $\hat{u}$  of a neural PDE solver, we seek a physically consistent solution  $u$  that satisfies the constraint  $h(u) = c$ . If consistency is defined in the  $L_2$  sense, then this can be formulated as finding the projection of  $\hat{u}$  into the feasible manifold,

$$\min_u \|u - \hat{u}\| \quad \text{s.t.} \quad h(u) = c. \quad (4.2)$$

**Nonlinear projection.** When  $h$  is nonlinear, there is no closed form solution to Equation 4.2. Instead, the problem can be formulated as an unconstrained minimization problem

$$\min_u \|u - \hat{u}\| + \lambda \|h(u) - c\|, \quad (4.3)$$

with weight  $\lambda \geq 0$ , which can be solved by numerical optimization. We use the LBFGS algorithm [18] throughout this work.

**Linear projection.** As Equation 4.2 admits analytical solutions when  $h$  is linear, an alternative is to linearize the constraints about  $\hat{u}$  and rearrange

$$h(u) \approx h(\hat{u}) + J_h(u - \hat{u}) \Rightarrow J_h u = c - h(\hat{u}) + J_h \hat{u}, \quad (4.4)$$

where  $J_h$  is the Jacobian of  $h$  computed at  $\hat{u}$ . We denote this linear system as  $Cu = b$ , where  $C = J_h$  and  $b = c - h(\hat{u}) + J_h \hat{u}$ . With linearized constraints, we formulate either the projection of  $\hat{u}$  onto the feasible set (Equation 4.5) or, if we suspect  $\hat{u}$  to be a poor approximation, the relaxed variant (Equation 4.6)

$$\min_u \|u - \hat{u}\| \quad \text{s.t.} \quad Cu = b \quad (4.5)$$

$$\min_u \|u - \hat{u}\| + \lambda \|Cu - b\|. \quad (4.6)$$

Both admit closed-form solutions [19]

$$u = \hat{u} - C^\top(CC^\top)^{-1}(C\hat{u} - b) \quad (4.7)$$

$$u = (I + \lambda C^\top C)^{-1}(\hat{u} + \lambda C^\top b). \quad (4.8)$$

The first is a projection enforcing the linearized constraints exactly when they are consistent, while the second trades constraint satisfaction and proximity to  $\hat{u}$  through  $\lambda \geq 0$ . Therefore, we call the first projection *constrained* and the second *relaxed*. Depending on the size of the system,  $C$  can be too large to fully materialize, and matrix inversion is known to be unstable [20–22]. In this case, instead of direct inversion, we solve the linear systems

$$CC^\top X = C\hat{u} - b, \quad (4.9)$$

$$(I + \lambda C^\top C)X = \hat{u} + \lambda C^\top b, \quad (4.10)$$

and respectively compute  $u = \hat{u} - C^\top X$  or  $u = X$ , using sparse solvers such as CG [23], GMRES [24], or BiCGSTAB [25]. These solvers only require access to the matrix-vector operators, so we leverage the fact that Equations 4.9 and 4.10 can be rewritten in terms of the JVP( $v$ ) :=  $v \mapsto Cv$  and VJP( $v$ ) :=  $v \mapsto C^\top v$  to obtain favorable scaling. Alternatively, for moderately sized systems the sparse matrix  $C$  can be used directly.

## 4.3 EXPERIMENTS

We investigate three dynamical systems: the 3D Lorenz ODE [26], the 1D Kuramoto–Sivashinsky (KS) PDE [27, 28], and the 2D Navier–Stokes (NS) PDE. More details on data generation are given in App. 4.A. In the KS and NS PDEs, we train a neural network to predict the full trajectory  $x_{1:T}$  from the initial state  $x_0$ , whereas in the

#### 4.4 CONCLUSION

|                                                                | Lorenz                    |                                | Kuramoto-Sivashinsky (64) |                                | Navier-Stokes (64)        |                                |
|----------------------------------------------------------------|---------------------------|--------------------------------|---------------------------|--------------------------------|---------------------------|--------------------------------|
|                                                                | MSE ( $\times 10^{-1}$ )↓ | Residual ( $\times 10^{-4}$ )↓ | MSE ( $\times 10^{-2}$ )↓ | Residual ( $\times 10^{-5}$ )↓ | MSE ( $\times 10^{-1}$ )↓ | Residual ( $\times 10^{-2}$ )↓ |
| <b>Data-driven (baseline = MLP for Lorenz, FNO for KS/NS)</b>  |                           |                                |                           |                                |                           |                                |
| Baseline                                                       | 7.78                      | 50.8                           | 4.01                      | 46.8                           | 13                        | 8.13                           |
| Baseline + Constrained                                         | <b>7.75</b>               | 13.1                           | 3.92                      | 16                             | 10.5                      | 4.21                           |
| Baseline + Relaxed                                             | 7.76                      | 13.3                           | 4                         | 39.9                           | 12.2                      | 3.4                            |
| Baseline + LBFGS                                               | 7.76                      | <b>1.18</b>                    | <b>3.91</b>               | <b>4.53</b>                    | <b>2.63</b>               | <b>0.00901</b>                 |
| <b>Physics-informed (PI = PINN for Lorenz, PINO for KS/NS)</b> |                           |                                |                           |                                |                           |                                |
| PI                                                             | 9.6                       | 51.7                           | 4.76                      | 4.27                           | 13.8                      | 6.48                           |
| PI + Constrained                                               | <b>9.57</b>               | 13.4                           | 4.77                      | 24.9                           | 11.5                      | 3.75                           |
| PI + Relaxed                                                   | 9.58                      | 13.5                           | 4.76                      | 3.05                           | 13                        | 2.94                           |
| PI + LBFGS                                                     | 9.58                      | <b>1.23</b>                    | <b>4.75</b>               | <b>1.27</b>                    | <b>3.21</b>               | <b>0.00956</b>                 |

Table 4.1. Results of the experiments. The evolution for KS and NS is reported on resolution 64. Lower is better. LBFGS achieves significantly lower constraint violation. In NS, where the solutions have small scale details, the MSE improvement is much more significant.

Lorenz system we give  $x_T$  as additional context to the network. For every equation we train two models: one that minimizes MSE only, and a physics-informed one that minimizes the MSE plus the MSE of the constraint loss  $h(u) - c$ . As physics-informed models, PINNs [8] are used for Lorenz, while PINOs [9] are used for KS and NS. We report test-set Mean Squared Error (MSE) and the squared  $L_2$  norm of the constraint violation  $\|h(u) - c\|$  (i.e. the residual) of the output of the networks without further processing, and after nonlinear and linear projections. For the NS equation, we show how the solution changes as the constraint violation reduces, and give a rough notion of the local constraint landscape. The results are shown in Table 4.1, with more details in App. 4.B.

**Lorenz.** An MLP and a PINN-MLP are trained on the Lorenz equation. For the relaxed projection,  $\lambda = 1000$  was used. All projections cut constraint violation by 70% or more and lower the MSE. The PINN-MLP has higher constraint violation despite having been trained to minimize it, as it did not train as effectively as the standard MLP under the same time and resource budget.

**Kuramoto-Sivashinsky.** A Fourier Neural Operator (FNO) [29] and a PINO were trained at a 64 resolution and evaluated at 64, 128, and 256 resolutions. Because KS dynamics are sufficiently resolved over these grids, the MSE is resolution-invariant, but constraint error rises with grid size. For the relaxed projection,  $\lambda = 10$  was used. Projections suppress the violation at all resolutions and also reduce MSE.

**Navier-Stokes.** Similarly to KS, an FNO and a PINO were trained at a 64 resolution and evaluated at 64, 128, and 256. For the relaxed projection,  $\lambda = \|\hat{u}\|$  was used. Across resolutions, the baseline MSE stays nearly constant because large-scale structures captured by the model dominate the error. Unlike KS, NS contains smaller-scale details, which shows up as constraint violations and missing fine structure in the model output. Once those violations are reduced, the MSE drops significantly, as shown in Figure 4.1 and Table 4.1. In Figure 4.2 we show the local structure of the constraint by measuring its violation along the LBFGS trajectory, helping explain why LBFGS outperforms the two linearization-based projections, as seen in Table 4.1. They are good locally at the beginning but quickly degrade, and close to the minimum the linear approximation becomes noisy.

#### 4.4 CONCLUSION

We demonstrated that training-free projection methods substantially outperform physics-informed training for enforcing governing equation constraints across multiple

#### 4.4 CONCLUSION

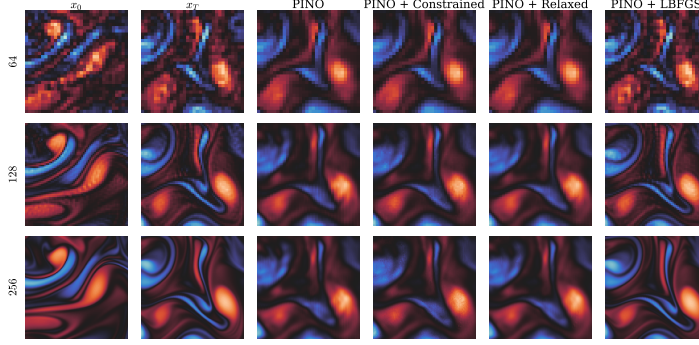


Figure 4.1. NS predictions comparing a PINO baseline with projection methods. Column 1 shows the initial condition  $x_0$ , Column 2 shows the ground truth  $x_T$ , and Columns 3-6 show predictions from PINO and projection methods. Only LBFGS successfully recovers the fine-scale structures; physics-informed models fail to capture details they never encountered during training.

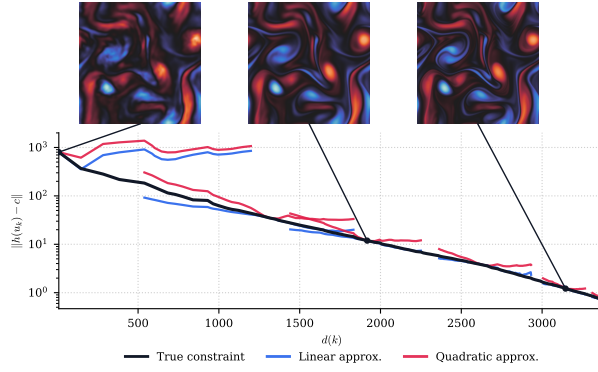


Figure 4.2. Constraint violation along LBFGS optimization path  $u_{0:200}$  from initial guess  $u_0 = \hat{u}$  to final solution  $u^* = u_{200}$  (black, log scale), where the  $x$ -axis measures the size  $d(k) = \sum_{i=0}^k \|u_{i+1} - u_i\|$  of the path  $u_{0:k}$ , for  $k = 0$  to 200. Blue and red curves show 1st and 2nd order Taylor approximations of the violation. The curves reveal two regimes: linear approximations work well initially but become inaccurate near the solution, while quadratic approximations remain reliable throughout, explaining why LBFGS outperforms linear projection methods.

#### 4.4 CONCLUSION

dynamical systems. LBFGS achieves near-perfect constraint satisfaction when the initial neural network output  $\hat{u}$  provides a reasonable approximation, while linearization-based methods offer computational efficiency at the cost of accuracy. Our analysis reveals that linear approximations are locally effective but degrade rapidly away from the linearization point, suggesting that iterative relinearization approaches, similar to Sequential Quadratic Programming [19], could improve performance if implemented efficiently.

Several challenges remain for broader adoption. Current projection methods, while computationally tractable, cannot be directly integrated as differentiable layers within neural networks due to the computational cost of backpropagation through the optimization process. Reducing projection overhead is essential for scaling to larger systems and enabling end-to-end training. Additionally, extending these techniques to generative models presents opportunities for improving temporal coherence by projecting sampled trajectories onto physically consistent manifolds. Future work should focus on developing differentiable projection operators and exploring their integration with modern generative architectures.

#### ACKNOWLEDGMENTS

This work was partially supported by Service Public de Wallonie Recherche under grant n° 2010235 – ARIAC by DIGITALWALLONIA4.AI.



## REFERENCES

- [1] Omer Rochman and Gilles Louppe. “Enforcing governing equation constraints in neural PDE solvers via training-free projections”. In *Machine Learning and the Physical Sciences Workshop (NeurIPS)*. 2025.
- [2] Hongkai Zheng et al. “InverseBench: Benchmarking Plug-and-Play Diffusion Priors for Inverse Problems in Physical Sciences”. Mar. 14, 2025. URL: <http://arxiv.org/abs/2503.11043> (visited on 04/29/2025). Pre-published.
- [3] Liuyi Chen et al. “Machine Learning Methods in Weather and Climate Applications: A Survey”. In *Applied Sciences* 13.21 (Jan. 2023), p. 12019.
- [4] Junzhe Yin et al. “Precipitation Nowcasting Using Physics Informed Discriminator Generative Models”. June 14, 2024. URL: <http://arxiv.org/abs/2406.10108> (visited on 08/14/2025). Pre-published.
- [5] Lukas Prantl et al. “Guaranteed Conservation of Momentum for Learning Particle-based Fluid Dynamics”. In *ArXiv* abs/2210.06036 (2022).
- [6] Margaux Boxho et al. “Turbulent Injection Assisted by Diffusion Models for Scale Resolving Simulations”. Aug. 6, 2025. URL: <http://arxiv.org/abs/2508.04318> (visited on 08/14/2025). Pre-published.
- [7] Mohsen Ahmadi et al. “Physics-Informed Machine Learning for Advancing Computational Medical Imaging: Integrating Data-Driven Approaches with Fundamental Physical Principles”. In *Artificial Intelligence Review* 58.10 (July 5, 2025), p. 297.
- [8] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations”. 2017.
- [9] Zongyi Li et al. “Physics-Informed Neural Operator for Learning Partial Differential Equations”. 2023.
- [10] Zihan Zhou, Xiaoxue Wang, and Tianshu Yu. “Generating Physical Dynamics under Priors”. Feb. 28, 2025. URL: <http://arxiv.org/abs/2409.00730> (visited on 06/24/2025). Pre-published.
- [11] Afla Ajithkumar Sophiya et al. “A comprehensive analysis of PINNs: Variants, Applications, and Challenges”. 2025.
- [12] M. Mattheakis et al. “Physical Symmetries Embedded in Neural Networks”. In *arXiv e-prints*, arXiv:1904.08991 (Apr. 2019), arXiv:1904.08991.
- [13] Gonzalo E. Constante-Flores, Hao Chen, and Can Li. “Enforcing Hard Linear Constraints in Deep Learning Models with Decision Rules”. 2025.
- [14] Jiahe Huang et al. “DiffusionPDE: Generative PDE-Solving Under Partial Observation”. Nov. 1, 2024. URL: <http://arxiv.org/abs/2406.17763> (visited on 06/24/2025). Pre-published.
- [15] Christian Jacobsen, Yilin Zhuang, and Karthik Duraisamy. “CoCoGen: Physically-Consistent and Conditioned Score-based Generative Models for Forward and Inverse Problems”. Oct. 20, 2024. URL: <http://arxiv.org/abs/2312.10527> (visited on 06/24/2025). Pre-published.
- [16] Giacomo Baldan et al. “Flow Matching Meets PDEs: A Unified Framework for Physics-Constrained Generation”. June 10, 2025. URL: <http://arxiv.org/abs/2506.08604> (visited on 06/24/2025). Pre-published.

- [17] Yulong Lu and Wuzhe Xu. “Generative Downscaling of PDE Solvers with Physics-Guided Diffusion Models”. Version 1. In (2024).
- [18] Dong C. Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In *Math. Program.* 45.1–3 (Aug. 1989), pp. 503–528.
- [19] Jorge Nocedal and Stephen J. Wright. “Numerical optimization”. 2. ed. Springer series in operations research and financial engineering. New York, NY: Springer, 2006. XXII, 664.
- [20] Nicholas J. Higham. “Accuracy and Stability of Numerical Algorithms”. Second. Society for Industrial and Applied Mathematics, 2002.
- [21] Lloyd N. Trefethen and David Bau. “Numerical Linear Algebra, Twenty-fifth Anniversary Edition”. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2022.
- [22] Gene H. Golub and Charles F. Van Loan. “Matrix Computations - 4th Edition”. Philadelphia, PA: Johns Hopkins University Press, 2013.
- [23] Magnus R. Hestenes and Eduard Stiefel. “Methods of conjugate gradients for solving linear systems”. In *Journal of research of the National Bureau of Standards* 49 (1952), pp. 409–435.
- [24] Youcef Saad and Martin H. Schultz. “GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems”. In *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869.
- [25] H. A. van der Vorst. “Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems”. In *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644.
- [26] Edward N. Lorenz. “Deterministic Nonperiodic Flow”. In *Journal of Atmospheric Sciences* 20.2 (1963), pp. 130–141.
- [27] Yoshiki Kuramoto. “Diffusion-Induced Chaos in Reaction Systems”. In *Progress of Theoretical Physics Supplement* 64 (Feb. 1978), pp. 346–367.
- [28] G.I. Sivashinsky. “Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations”. In *Acta Astronautica* 4.11 (1977), pp. 1177–1206.
- [29] Zongyi Li et al. “Fourier Neural Operator for Parametric Partial Differential Equations”. In *International Conference on Learning Representations*. 2021.

## 4.A DATA

### 4.A.1 LORENZ DATA

The Lorenz dataset is generated by integrating the 3-D Lorenz-63 [26] system

$$\dot{x} = \sigma(y - x), \quad \dot{y} = x(\rho - z) - y, \quad \dot{z} = xy - \beta z. \quad (4.11)$$

We use  $\sigma = 10$ ,  $\rho = 28$ , and  $\beta = 8/3$ . Trajectories are generated on-the-fly with  $x_0 \sim \mathcal{N}(0, 0.1)$ . Euler integration was used with a fixed step  $dt = 1/512$  for  $T = 512$  steps, giving a time horizon  $t_{\max} = 1$ . Independent random seeds create disjoint training, validation, and test splits.

### 4.A.2 KURAMOTO-SIVASHINSKY DATA

The 1-D Kuramoto–Sivashinsky equation [27, 28]

$$\partial_t u + \partial_{xx} u + \partial_{xxxx} u + u \partial_x u = 0 \quad (4.12)$$

is integrated on the periodic domain  $x \in [0, 64]$  with a pseudospectral solver. Spatial derivatives use the FFT on grids with  $N \in \{64, 128, 256\}$  points and no de-aliasing. Time stepping was done using the first-order backward-difference formula (BDF1) with fixed step  $dt = 10^{-1}$  for 512 steps, giving  $t_{\max} = 51.2$ . We generated 65,536 train, 128 validation, and 128 test simulations. During training, we used a subset of each solution, starting at step 256 and ending at step 512, corresponding to  $t \in (25.6, 51.2)$ . Each implicit step is solved by Newton iterations. Initial data are random sums of ten cosine modes,

$$u_0(x) = \sum_{k=1}^{10} a_k \cos(2\pi\omega_k x/64 + \phi_k), \quad (4.13)$$

where amplitudes  $a_k \sim \mathcal{U}(0, 1)$ , frequencies  $\omega_k \sim \mathcal{U}(1, 5)$  and phases  $\phi_k \sim \mathcal{U}(0, 2\pi)$  are sampled independently. The dataset comprises 65,536 training, 128 validation, and 128 test simulations (total 65,792), each generated with an independent initial condition and solver seed.

#### 4.A.3 NAVIER-STOKES DATA

The 2-D incompressible Navier–Stokes equations in vorticity form

$$\begin{aligned} \frac{\partial \omega}{\partial t} &= -\mathbf{u} \cdot \nabla \omega + \frac{1}{\text{Re}} \Delta \omega + f, \\ \mathbf{u} &= \left( \frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right), \\ \Delta \psi &= \omega, \end{aligned} \quad (4.14)$$

with the forcing  $f(\omega; x, y, t) = -4 \cos(4y) - 0.1\omega(x, y, t)$  are integrated on the periodic domain  $[0, 2\pi]^2$  with  $\text{Re} = 1000$  using a pseudospectral solver. Spatial derivatives use the FFT on grids with  $N \in \{64, 128, 256\}$  and 2/3 de-aliasing. Time stepping uses the explicit Heun method with fixed step  $dt = 2 \times 10^{-3}$ . We record 256 snapshots at uniform intervals, giving a time horizon  $T = 8.192$ . Initial data are zero-mean Gaussian random fields sampled at the highest resolution and projected to each grid. We generated 4096 training, 128 validation, and 128 test simulations.

## 4.B ADDITIONAL RESULTS

|                        | MSE ( $\times 10^{-1}$ ) | Residual ( $\times 10^{-4}$ ) |
|------------------------|--------------------------|-------------------------------|
| MLP                    | 7.78                     | 50.8                          |
| MLP + Constrained      | <b>7.75</b>              | 13.1                          |
| MLP + LBFGS            | 7.76                     | <b>1.18</b>                   |
| MLP + Relaxed          | 7.76                     | 13.3                          |
| MLP-PINN               | 9.6                      | 51.7                          |
| MLP-PINN + Constrained | <b>9.57</b>              | 13.4                          |
| MLP-PINN + LBFGS       | 9.58                     | <b>1.23</b>                   |
| MLP-PINN + Relaxed     | 9.58                     | 13.5                          |

Table 4.2. Lorenz results. (Left) SE between GTs and generated trajectories, averaged over all trajectories and timesteps. (Right) Mean squared  $L_2$  norm of the constraint violation of generated trajectories.

|                    | MSE ( $\times 10^{-2}$ ) |             |             | Residual ( $\times 10^{-5}$ ) |             |             |
|--------------------|--------------------------|-------------|-------------|-------------------------------|-------------|-------------|
|                    | 64                       | 128         | 256         | 64                            | 128         | 256         |
| FNO                | 4.01                     | 3.97        | 3.98        | 46.8                          | 267         | 273         |
| FNO + Constrained  | 3.92                     | 3.72        | 3.73        | 16                            | 71.4        | 74.6        |
| FNO + LBFGS        | <b>3.91</b>              | <b>3.65</b> | <b>3.66</b> | <b>4.53</b>                   | <b>3.73</b> | <b>3.77</b> |
| FNO + Relaxed      | 4                        | 3.87        | 3.88        | 39.9                          | 182         | 185         |
| PINO               | 4.76                     | 5.56        | 5.55        | 4.27                          | 874         | 896         |
| PINO + Constrained | 4.77                     | 5.42        | 5.01        | 24.9                          | 368         | 239         |
| PINO + LBFGS       | <b>4.75</b>              | <b>4.67</b> | <b>4.66</b> | <b>1.27</b>                   | <b>1.11</b> | <b>1.18</b> |
| PINO + Relaxed     | 4.76                     | 5.18        | 5.17        | 3.05                          | 502         | 509         |

Table 4.3. Results for Kuramoto-Sivashinsky. (Left) SE between GTs and generated trajectories, averaged over all trajectories and timesteps. (Right) Mean squared  $L_2$  norm of the constraint violation of generated trajectories. Constraint error increases with the resolution unless projection is performed.

|                    | MSE ( $\times 10^{-1}$ ) |             |             | Residual ( $\times 10^{-2}$ ) |                |                |
|--------------------|--------------------------|-------------|-------------|-------------------------------|----------------|----------------|
|                    | 64                       | 128         | 256         | 64                            | 128            | 256            |
| FNO                | 13                       | 9.09        | 8.85        | 8.13                          | 6.58           | 6.53           |
| FNO + Constrained  | 10.5                     | 7.4         | 7.36        | 4.21                          | 3.1            | 2.97           |
| FNO + LBFGS        | <b>2.63</b>              | <b>2.52</b> | <b>2.58</b> | <b>0.00901</b>                | <b>0.00793</b> | <b>0.00813</b> |
| FNO + Relaxed      | 12.2                     | 8.41        | 8.18        | 3.4                           | 2.9            | 2.88           |
| PINO               | 13.8                     | 9.91        | 9.61        | 6.48                          | 6.17           | 6.13           |
| PINO + Constrained | 11.5                     | 8.27        | 8.13        | 3.75                          | 3.07           | 2.97           |
| PINO + LBFGS       | <b>3.21</b>              | <b>3.01</b> | <b>3.02</b> | <b>0.00956</b>                | <b>0.00829</b> | <b>0.00832</b> |
| PINO + Relaxed     | 13                       | 9.24        | 8.95        | 2.94                          | 2.76           | 2.74           |

Table 4.4. Results for Navier-Stokes. (Left) SE between GTs and generated trajectories, averaged over all trajectories and timesteps. (Right) Mean squared  $L_2$  norm of the constraint violation of generated trajectories.

## 5 PREDICT-PROJECT-RENOISE: SAMPLING DIFFUSION MODELS UNDER HARD CONSTRAINTS

*Por una línea razonable o una recta noticia hay leguas de insensatas cacofonías,  
de fárragos verbales y de incoherencias.*

— Borges, *La biblioteca de Babel*

### ADDENDUM

This chapter is based on the paper:

Omer Rochman-Sharabi and Gilles Louppe. “Predict-Project-Renoise: Sampling Diffusion Models under Hard Constraints”. 2026. Submitted

This chapter is the work of Omer as lead author, who formulated the method, built the implementation, carried out the experiments, and drafted the manuscript. Gilles supervised the project and contributed to the writing.

### ABSTRACT

Diffusion models cannot enforce hard constraints, yet applications in the physical sciences demand exact satisfaction of conservation laws, boundary conditions, and observational consistency. In this work, we identify a corrector kernel whose unique stationary distribution is the constrained marginal at each noise level, and approximate it by iteratively projecting through the denoiser and renoising via the forward kernel. The resulting Predict-Project-Renoise (PPR) algorithm enables sampling from pretrained diffusion models under hard constraints. Its three components are each necessary: projecting through the denoiser keeps samples close to the data manifold, while renoising and iterating drive samples toward the constrained marginal. On 2D distributions, the Kuramoto-Sivashinsky equation, and global weather forecasting with a  $10^8$ -dimensional atmospheric model, PPR simultaneously achieves low constraint violations and high distributional fidelity, a combination that existing methods fail to deliver.

### 5.1 INTRODUCTION

Diffusion models [2–4] have emerged as the state-of-the-art framework for deep generative modeling, showing strong performance in approximating high-dimensional data distributions  $p$  across modalities. However, in many scientific and engineering

domains, such as medical imaging, fluid dynamics, or climate modeling, sampling  $p$  is insufficient. Samples  $x$  must also satisfy hard constraints  $x \in C$ , such as exact consistency with observations or governing equations, zero divergence in incompressible fluids, conservation laws, boundary and initial conditions, or geometric constraints.

Although training-based approaches can incorporate constraints [5–7], they often lack flexibility, requiring retraining for different constraints, and enforce the constraints only softly. For these reasons, recent efforts focus on *training-free constrained sampling*, where a pre-trained unconditional model is guided to satisfy constraints during inference. Current methods typically rely on treating constraints as observations [8] or implement projection-based corrections [9, 10].

When treating the constraint as an observation, Bayesian decomposition can be used to express the posterior score as a prior score plus a likelihood score. While effective for noisy inverse problems [11], especially in the context of images, these methods are ill-suited for hard constraints: the implicit likelihood approaches a Dirac delta, causing gradients to become unstable and informative only in the immediate vicinity of the feasible set  $C$ . Projection methods can enforce feasibility explicitly, but have theoretical and practical pitfalls as they do not correctly sample the true constrained distribution. This manifests as three problems: (i) the constraints are violated, (ii) the constrained distribution is incorrect, and (iii) the sample lies outside of the support  $S$  of the data distribution  $p$ .

For nonlinear constraints, the projection typically has no closed-form solution, so it must be approximated with an iterative solver. Moreover, it is often unclear what exactly should be projected. Directly projecting the noisy state onto the constraint set can be ill-posed or numerically unstable, since many constraints are only meaningful for clean data. Projecting an estimate of the clean sample is not a fix either. Sample consistency with  $p$  is often broken, for example, by introducing discontinuities into otherwise continuous samples. Finally, repeated projection steps can systematically bias the sampler. In the extreme, this behaves like an attractor toward “nearest” feasible points, causing mode collapse rather than sampling from the constrained distribution.

**Contributions.** In this work, we address these limitations by arguing that correct constrained sampling requires approximating the constrained marginal distributions  $p_t^C$  induced by a forward diffusion SDE starting from the constrained set  $C$ . We define the constrained forward process and derive the associated constrained marginals and backward process. Based on the backward process, we identify a corrector kernel that has the constrained marginals as its unique stationary distributions, design the Predict-Project-Renoise (PPR) algorithm that approximates that kernel, and show that each of its components is necessary. PPR achieves higher feasibility rates, lower constraint violations, and better mass distribution compared to other methods.

## 5.2 SCORE-BASED GENERATIVE MODELING

**Forward process.** Generative modeling seeks to approximate a data distribution  $p(x)$  of  $x \in \mathbb{R}^D$ . In score-based diffusion [4], this is done by learning to reverse a forward corruption process. Following Lai et al. [12], we define the *forward process* as the stochastic differential equation (SDE)

$$dx_t = f(x_t, t)dt + g(t)dw_t, \quad (5.1)$$

where  $f : \mathbb{R}^{D+1} \rightarrow \mathbb{R}^D$  is a drift term,  $g : \mathbb{R} \rightarrow \mathbb{R}$  is a scalar diffusion coefficient, and  $w_t$  is a standard Wiener process. Starting from  $p_0 = p$ , this SDE defines a series of

marginal densities  $\{p_t(x_t)\}_{t \in [0, T]}$  which interpolate between the data distribution  $p_0$  and  $p_T$ . Choosing  $f$  and  $g$  defines the perturbation or *forward kernels*  $p(x_t|x_0)$ , inducing the marginals

$$p_t(x_t) = \int p(x_t | x_0) p(x_0) dx_0. \quad (5.2)$$

If  $f$  and  $g$  are chosen appropriately, then  $p(x_t|x_0)$  and  $p_T$  become Gaussian [13]. In particular,  $p_T$  can become  $\mathcal{N}(0, \sigma_T I)$ .

**Backward process.** Anderson [14] shows that given a forward process (Eq. 5.1), there exists a *reverse* or *backward process* from  $T$  to 0 transporting  $p_T$  back to  $p_0$  given by

$$dx_t = [f(x_t, t) - g^2(t) \nabla_{x_t} \log p_t(x_t)] dt + g(t) d\tilde{w}_t, \quad (5.3)$$

where  $x_T \sim p_T(x_T)$  and  $\tilde{w}_t$  is the reverse-time Wiener process. In this reverse process, the marginal distributions of  $x_t$  are identical to those of the forward process. This allows for sampling from  $p$  by generating a sample from  $p_T$  and then simulating the reverse process.

**Training and sampling.** Since the score  $s(x_t) = \nabla_{x_t} \log p_t(x_t)$  is unknown, it has to be learned from data. We follow Karras et al. [15] and parameterize the score using Tweedie’s formula  $s_\theta(x_t) = (d_\theta(x_t) - x_t)/\sigma_t^2$  [16, 17]. The resulting denoising objective minimizes

$$\mathbb{E}_{x_0 \sim p(x_0)} \mathbb{E}_{x_t \sim p(x_t|x_0)} \|d_\theta(x_t) - x_0\|^2. \quad (5.4)$$

The denoiser  $d_\theta(x_t) \approx \mathbb{E}[x_0|x_t]$  predicts the posterior mean of the clean sample given the noisy input. Once it is available, a way to sample from the reverse process is to use the predictor-corrector (PC) sampler (Song et al. [4], Algorithm 4). This algorithm alternates between a PREDICT step (e.g. DDPM [3], DDIM [18]), which moves the samples from  $p_{t+1}$  to  $p_t$ , and a number of CORRECT steps (e.g. Langevin steps based on the score). The CORRECT steps reduce the discretization error by pulling samples toward higher-density regions of  $p_t$  and re-injecting noise to increase diversity.

## 5.3 METHOD

The reverse process (Eq. 5.3) allows us to approximate the data distribution  $p$ , but in many domains we are interested in imposing constraints on the samples. Let  $S \subseteq \mathbb{R}^D$  be the support of  $p(x)$  and  $C \subseteq \mathbb{R}^D$  be a (constrained or feasible) set corresponding to samples with certain characteristics, such as conservation laws, adherence to given dynamics, or a desired decay in the energy spectrum.  $C$  can be represented using a nonnegative constraint function  $c(x) \geq 0$  as  $C := \{x \in \mathbb{R}^D : c(x) = 0\}$ . The intersection  $S \cap C$  strictly restricts the prior to the feasible set, inducing the density  $p^C \propto p(x_0) \mathbb{1}_C(x_0)$ , where  $\mathbb{1}_C(x)$  is the indicator function of  $C$ . We say a sample  $x \in S \cap C$  is *feasible*.

### 5.3.1 CONSTRAINED DIFFUSION

**Constrained forward process.** We define the *constrained forward process* similarly to Eq. 5.1, but with  $x_0 \sim p^C(x_0)$ . This constrained process induces a sequence of marginals  $\{p_t^C\}_{t \in [0, T]}$ , given by

$$p_t^C(x_t) \propto \int p(x_t | x_0) p(x_0) \mathbb{1}_C(x_0) dx_0. \quad (5.5)$$

The unconstrained and constrained forward processes share the same dynamics and transition kernels but their different initial densities lead to distinct marginals and reverse processes.

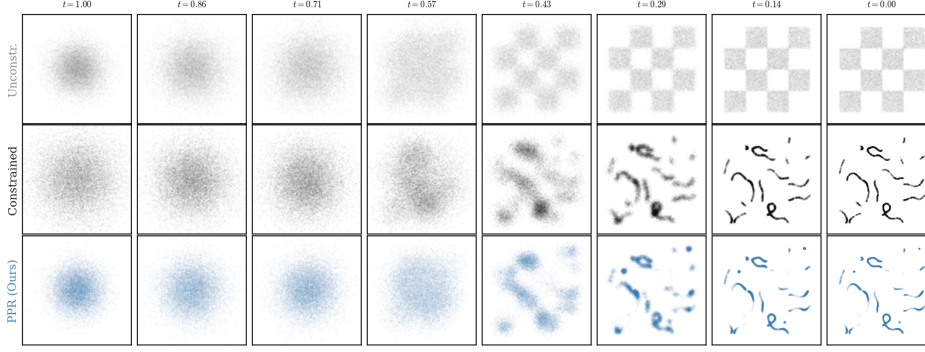


Figure 5.1. Visualization of the constrained process on DATA2D, showing the target constrained marginals  $p_t^C$ , the unconstrained marginals  $p_t$ , and the marginals induced by PPR.

**Constrained backward process.** Mirroring the unconstrained backward process in Eq. 5.3, the *constrained reverse process* is

$$dx_t = [f(x_t, t) - g^2(t)\nabla_{x_t} \log p_t^C(x_t)] dt + g(t)d\tilde{w}_t. \quad (5.6)$$

This process involves the constrained score function  $\nabla_{x_t} \log p_t^C(x_t)$ . It can be computed by taking the log and then the gradient of the constrained marginals (Eq. 5.5)

$$\nabla_{x_t} \log \int_C p(x_t|x_0)p(x_0)dx_0, \quad (5.7)$$

dropping the normalization constant because it vanishes under the derivative and absorbing the indicator function into the integral. Then, from Bayes' theorem,  $p(x_t|x_0)p(x_0) = p(x_0|x_t)p(x_t)$ , we get

$$\nabla_{x_t} \log p_t^C(x_t) = \nabla_{x_t} \log p_t(x_t) + \nabla_{x_t} \log \int_C p(x_0|x_t)dx_0. \quad (5.8)$$

The constrained score is thus the unconstrained score plus the gradient of the log-feasibility  $\nabla_{x_t} \log \int_C p(x_0|x_t)dx_0 = \nabla_{x_t} \log \mathbb{P}(x_0 \in C|x_t)$ , a term pointing in the direction (in  $x_t$ -space) of increasing probability of  $x_0$  being in  $C$ . Figure 5.1 gives visual intuition on these processes.

### 5.3.2 PREDICT-PROJECT-RENOISE (PPR) SAMPLING

At each reverse step, the unconstrained predictor (Algorithm 4) provides  $x_t \sim p_t$ , whose denoised estimates are unlikely to satisfy the constraints exactly. Based on the structure of the constrained backward process (Section 5.3.1), we identify the following corrector:

**Proposition 1.** Let  $K_t(x'_t|x_t) = \int_C p(x'_t|x_0)p^C(x_0|x_t)dx_0$  with  $p^C(x_0|x_t) = \frac{p(x_0|x_t)\mathbb{1}_C(x_0)}{\mathbb{P}(x_0 \in C|x_t)}$ . Under this kernel,  $p_t^C$  is the unique stationary distribution and  $(K_t)^M q \rightarrow p_t^C$  as  $M \rightarrow \infty$  for any initial distribution  $q$ . See Appendix 5.C for the proof.

That is, iterating the kernel  $K_t$  at a fixed noise level acts as a corrector that drives any sample, including  $x_t \sim p_t$  from the predictor, toward the constrained marginal  $p_t^C$ , just as Langevin correction drives samples toward  $p_t$ . Its structure is a two-step conditional resampling: sample a feasible  $x_0 \in C$  from the constrained posterior  $p^C(x_0|x_t)$ , then renoise via the forward kernel to obtain  $x'_t$ . Renoising is essential as it restores the correct noise level and injects the stochasticity needed for mixing.



Of course, sampling from the constrained posterior  $p^C(x_0|x_t)$  is itself intractable. Instead, PPR approximates it with a point estimate  $p^C(x_0|x_t) \approx \delta(x_0 - d_\theta(\text{PROJECT}(x_t)))$ , where  $\text{PROJECT}(x_t) := \arg \min_x c(d_\theta(x))$  is solved via an off-the-shelf optimizer initialized at  $x_t$ . Plugging this approximation into  $K_t$  yields

$$K_t^{\text{PPR}}(x'_t|x_t) := p(x'_t | d_\theta(\text{PROJECT}(x_t))), \quad (5.9)$$

where the integral over  $C$  in  $K_t$  collapses to a single forward kernel evaluation at the projected point estimate. Again, the renoising step is critical: without it,  $K_t^{\text{PPR}}$  reduces to a deterministic map that collapses to fixed points of the projection; with it, the kernel retains the stochasticity needed for mixing. Following Proposition 1, we iterate  $K_t^{\text{PPR}}$   $M$  times to drive samples toward  $p_t^C$ . In summary, the proposed *Predict-Project-Renoise* sampler (Algorithm 5) alternates at each reverse step between a predictor update, a projection through the denoiser, and a renoising step, with the last two repeated  $M$  times.

**Approximation error.** Replacing the constrained posterior with a point estimate introduces an error that we characterize in Appendix 5.C. The error decomposes into two terms: a mixing term that decays exponentially with  $M$ , justifying the repetition of project-renoise steps, and an irreducible bias from the point estimate approximation, which is small when the posterior concentrates around a single feasible  $x_0$ .

**Projecting through the denoiser.** There are three natural ways to enforce feasibility during reverse sampling. The first is to minimize  $c(x_t)$  with respect to  $x_t$  directly. However, this is problematic because constraints are defined on clean data, while  $x_t \notin S$  almost surely, making  $c(x_t)$  ill-conditioned or undefined. The second is to compute  $x_0 = d_\theta(x_t)$  and minimize  $c(x_0)$  with respect to  $x_0$ . This evaluates the constraint where it is meaningful, but the optimized  $x_0$  is free to leave  $S$ , producing artifacts such as discontinuities (Section 5.5). PPR takes a third route: minimize  $c(d_\theta(x_t))$  with respect to  $x_t$ . Because the denoiser is trained to output elements of  $S$ , the projected estimate  $d_\theta(x_t^*)$  is encouraged to remain on the data manifold. Backpropagation through  $d_\theta$  further propagates local constraint information through learned correlations, helping maintain  $x_0 \in C \cap S$  globally. The cost is higher due to differentiation through  $d_\theta$ , but can be negligible when the constraint itself dominates computation, as in latent-space models with large decoders. In practice, **PROJECT** is solved with standard optimizers (e.g., L-BFGS [19], Adam [20]), and the projection budget provides a direct tradeoff between compute and sample feasibility.

---

**Algorithm 4** Predictor-corrector sampling.

---

```

1 Input: denoiser  $d_\theta$ , predictor PREDICT,
  corrector CORRECT, and  $N$ .
2 Initialize: sample  $x_T \sim p_T(x_T)$ 
3 for decreasing  $t \in \{T-1, \dots, 0\}$  do
4    $\hat{x}_0 \leftarrow d_\theta(x_{t+1})$ 
5    $x_t \leftarrow \text{PREDICT}(x_{t+1}, \hat{x}_0, t+1)$ 
6   for  $i = 1, \dots, N$  do
7      $\hat{x}_0 \leftarrow d_\theta(x_t)$ 
8      $x_t \leftarrow \text{CORRECT}(x_t, \hat{x}_0, t)$ 
9 Output:  $x_0$ 
```

---



---

**Algorithm 5** Predict-Project-Renoise.

---

```

1 Input: denoiser  $d_\theta$ , predictor PREDICT,
  projection  $\Pi_d$ , steps  $M$ .
2 Initialize: sample  $x_T \sim p_T(x_T)$ 
3 for decreasing  $t \in \{T-1, \dots, 0\}$  do
4    $\hat{x}_0 \leftarrow d_\theta(x_{t+1})$ 
5    $x_t \leftarrow \text{PREDICT}(x_{t+1}, \hat{x}_0, t+1)$ 
6   for  $i = 1, \dots, M$  do
7      $x_t^* \leftarrow \text{PROJECT}(x_t)$ 
8      $x_t \sim p(x_t | x_0 = d_\theta(x_t^*))$ 
9 Output:  $x_0$ 
```

---

## 5.4 RELATED WORK

Previous work on constrained sampling in diffusion models is summarized in Table 5.1. First, we focus on projection-based constrained samplers and then discuss a related body of work on posterior sampling under noisy observation processes. To help understand how previous methods relate to each other, we annotate each projection-based method according to the space in which the projection is done. If done with respect to  $x_t$ , we say the projection is in  $x_t$ -space and use red. If done with respect to  $x_0$ , we call it  $x_0$ -space and use blue. We will also add “through the denoiser” if the  $x_t$ -space projection is done using  $c(d_\theta(x_t))$  and “renoise” if any noise is added after the projection.

### Projection-based constrained sampling.

A number of methods enforce constraints by modifying reverse-time sampling with projections. Christopher et al. [27] ( $x_t$ -space) and Liu et al. [29] focus on convex constraints, while Jayaram et al. [28, 30] study linear constraints. Both are  $x_t$ -space and the latter project through the denoiser.

Zampini et al. [25] ( $x_t$ -space) use proximal Langevin dynamics within the latent  $x_t$  space, propagating gradients through the decoder. Their approach has the side effect of mitigating potential deviations from  $S$  by leveraging a structured latent space where the decoder can correct samples. In contrast, Narasimhan et al. [24] ( $x_0$ -space) operate solely in  $x_0$ -space, which can cause samples to be inconsistent with the data distribution, i.e.  $x_0 \notin S$ . For example, discontinuities can appear in continuous data (see, in their appendix, Figures 6 and 10). Adding more information via constraints can mask this issue. Huang et al. [26] ( $x_t$ -space through the denoiser with renoising) propose a heuristic trust region method based on the magnitude of the predicted noise  $\epsilon_\theta \propto x_t - d_\theta(x_t)$ . We refer to this method as TS for Trust Sampling. Christopher et al. [31] ( $x_0$ -space) uses domain knowledge, global constraints, and the forward kernel for protein design. In the flow matching context, Cheng et al. [32] ( $x_0$ -space) use extrapolation, a known exact correction, and interpolation to generate the constrained samples.

Table 5.1. Summary of related works, showing in which space they operate, if they renoise, and whether they propagate information through  $d_\theta$ .

| Method                             | Space | Renoise | Prop. $d_\theta$ |
|------------------------------------|-------|---------|------------------|
| <i>Constraints as observations</i> |       |         |                  |
| DPS [21]                           | $x_t$ | ✗       | ✓                |
| MMPS [22]                          | $x_t$ | ✗       | ✓                |
| ReSample/HDC [23]                  | $x_0$ | ✓       | ✗                |
| <i>Constrained sampling</i>        |       |         |                  |
| CPS [24]                           | $x_0$ | ✗       | ✗                |
| Zampini et al. [25]                | $x_t$ | ✗       | ✗                |
| TS [26]                            | $x_t$ | ✓       | ✓                |
| PDM [27]                           | $x_t$ | ✗       | ✗                |
| CDIM [28]                          | $x_t$ | ✗       | ✗                |
| <b>PPR (Ours)</b>                  | $x_t$ | ✓       | ✓                |

**Posterior sampling under observation processes.** A distinct yet related body of work focuses on sampling from posteriors derived from observation processes. While constrained sampling enforces strict adherence to a feasible set, observation-based methods typically condition generation on external measurements, often assuming an underlying Gaussian likelihood  $p(y|x) = \mathcal{N}(y | A(x), \sigma^2)$ , where  $A$  is the observation process and  $y$  is the observation value, as proposed in [21, 22, 33–35]. Theoretically, hard constraints can be framed as a special case of inverse problems by modeling the observation likelihood as a Dirac delta distribution  $\mathcal{N}(y | A(x), \sigma^2) \xrightarrow{\sigma^2 \rightarrow 0} \delta(y - A(x))$ . However, applying standard posterior sampling algorithms [11, 36] in this regime presents challenges, as these methods are generally designed for more informative likelihoods rather than the narrow distributions required for strict constraint satisfaction. Notably among these, Song et al. [23] ( $x_0$ -space) focus on constraints and do renoising with a hand-designed kernel.

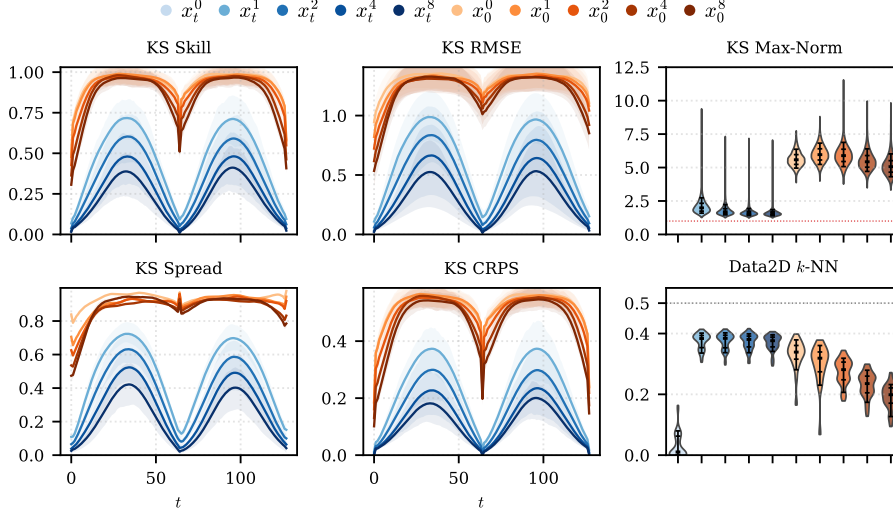


Figure 5.2. KS and DATA2D PPR ablation. We vary the number of kernel applications  $M$  and whether information is propagated through the denoiser. Projecting in  $x_0$ -space rather than  $x_t$ -space worsens the constrained distribution for both datasets and in KS,  $x_0$ -space samples are not in  $S$  because they are discontinuous. When projecting through the denoiser in  $x_t$ -space, larger  $M$  improves the constrained distributions in both settings. For KS, skill, RMSE, and CRPS measure prediction accuracy; for DATA2D,  $k$ -NN measures constrained-distribution quality. Max-Norm is better when closer to 1 (dotted red line). In the KS metrics, the x-axis  $t$  denotes system time, not diffusion time.

## 5.5 EXPERIMENTS

**Protocol.** We consider three experimental settings: DATA2D, Kuramoto-Sivashinsky (KS), and WEATHER. DATA2D comprises 2D problems where the ground truth constrained distribution can be approximated. KS is based on the 1D Kuramoto-Sivashinsky equation [37] and assesses our method and the baselines in a high-dimensional setting. Finally, WEATHER uses Appa, a global latent weather model [38], to test PPR in a large-scale, realistic setting. The full numerical results, additional experiments, and qualitative samples are shown in the Appendices 5.D, 5.E, 5.F, and 5.G. Numerically, we consider  $x \in C$  if  $c(x) \leq 4 \times 10^{-4}$  for KS and  $c(x) \leq 4 \times 10^{-6}$  for DATA2D.

**DATA2D.** We first train diffusion models on two distinct 2D distributions, a uniform checkerboard (Fig. 5.1, top), and a mixture of Gaussians, which serve as priors. We then generate 12 random functions  $f(x)$  by sampling cosine series. For each function, we define the constraint  $c(x) = 1 - e^{-f(x)^2}$  and approximate the ground-truth (GT) constrained distribution by rejection sampling from the diffusion model prior with a small tolerance  $\epsilon$ , that is,  $\text{GT} = \{x \in S : c(x) \leq \epsilon\}$ . The accepted samples are then refined with a few deterministic gradient steps to better satisfy the constraint. This results in 12 constraints for each of the 2 distributions, giving 24 total experimental settings. For each of these, we generated ensembles with PPR and the baselines to approximate the GT constrained distribution. More details can be found in Appendix 5.B. To assess the quality of the resulting constrained distributions, we use the  $k$ -nearest-neighbor ( $k$ -NN) cross-edge rate [39]. The  $k$ -NN cross-edge rate constructs a graph on the union of GT and model samples, and counts the number of edges that connect a GT sample to a model sample. The cross-edge rate (cross edges divided by total edges) is expected to be 0.5 when the two distributions match. The results are averaged across all prior distributions and all 12 constraints.

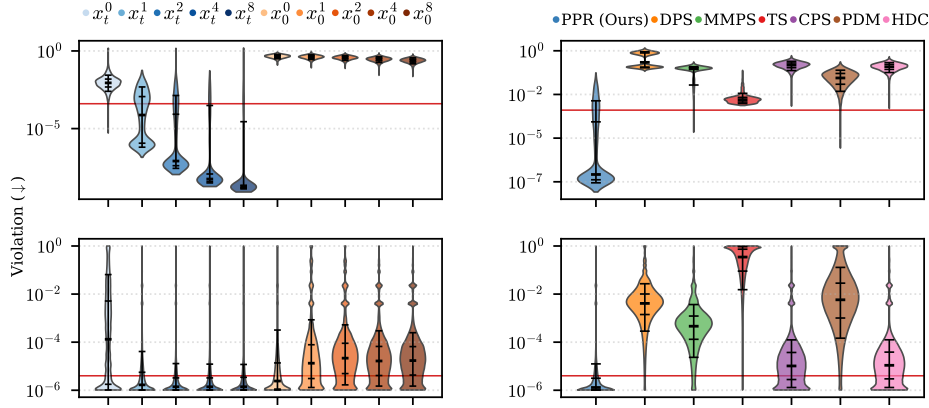
**KURAMOTO-SIVASHINSKY.** We train a diffusion prior to generate trajectories of the 1D KS equation  $u_t = -u_{xx} - u_{xxxx} - uu_x$ . To avoid confusion with diffusion time, we denote the dynamical system time with a superscript and generated samples with a subscript  $u_0$ , which is omitted for the ground truth. We generate a dataset by sampling cosine series initial conditions  $u^0 \in \mathbb{R}^{512}$  and solving the equation with periodic boundary conditions for  $t \in [0, L]$ , resulting in trajectories  $u^{1:1024}$ . We then select a subset of each trajectory  $u^{512:1024} \in \mathbb{R}^{512 \times 512}$ . This is done to ensure that the system has reached the statistically stationary regime. To form the dataset, we downsize each trajectory to  $\mathbb{R}^{128 \times 128}$ . As evaluating the quality of approximations of conditional distributions for high-dimensional systems is an open problem [40], we rely on proxy quantities. To test the distribution of samples, we frame the perfect nonlinear observation  $c(u_0) = \sum_i \|\sin(u_0^i) - \sin(u^i)\|^2$ ,  $i \in \{0, 64, 127\}$  as a constraint. In Appendix 5.F, we test other constraints. We observe each of the 256 test trajectories and for each method we generate an ensemble of 32 trajectories. The system is deterministic, so we expect to observe close agreement with the trajectory the observation was obtained from, i.e., a delta distribution. To judge the quality of the constrained distribution, we present 4 metrics that are best considered together. The root-mean-squared-error (RMSE) measures how far members of the ensemble are from the ground truth, on average. The skill measures how far the ensemble mean is from the ground truth, that is, the bias of the ensemble. The spread measures the deviation of the ensemble from said mean, i.e., how close it is to the delta. Lastly, the continuous ranked probability score (CRPS) penalizes both bias and misrepresentation of uncertainty (too much or too little spread). See Appendix 5.A for their definitions.

**PPR ablation.** Proposition 1 indicates that renoising and repeated kernel application are fundamental requirements in approximating the constrained marginals and we argue that using the denoiser in the projection loop has key benefits. We justify the claims on KS and DATA2D by carefully ablating each component and showing that without any one of them performance degrades dramatically. To that end, we introduce a PPR variant that does not use the denoiser in the projection. In the variant, labeled  $x_0^M$  in the figures, we project  $x_0$  directly ( $x_0$ -space), without the denoiser, which is only used to get the initial  $x_0 = d_\theta(x_t)$ . That makes the projection  $\Pi(x_0) = \arg \min_x c(x)$  and the transition kernel  $K_t^{\text{var}}(x'_t|x_t) = p(x'_t|\Pi(x_0))$ . For PPR and the variant, we study no renoising ( $M = 0$ ), only renoising ( $M = 1$ ) and renoising plus reapplication of the  $K_t^{\text{PPR}}$  kernel ( $M > 1$ ).

We run the ablations on KS and DATA2D, confirming the necessity of the projection through the denoiser, renoising, and multiple kernel applications. Figures 5.2 and 5.3a show that not using the denoiser (in orange) results in samples that score worse on all metrics and have much higher constraint violation ( $x_0 \notin C$ ), with the number of kernel applications doing little to improve the situation. The Max-Norm shows that the KS samples are not in  $S$  as they present sharp discontinuities. The advantage of the denoiser (in blue) is clear; the samples remain in  $S$  while improving all metrics and the constraint violation, achieving  $x_0 \in C$ . Increasing the number of kernel applications  $M$  consistently improves the samples at a linear computational cost. No renoising, i.e.  $M = 0$  is omitted from the KS subfigures as it diverged, supporting the necessity of  $M > 0$ . These conclusions are confirmed on DATA2D.  $x_t$ -space variants achieve  $k$ -NN cross-edge rates closer to 0.5 and lower constraint violation.  $M = 0$  fails catastrophically but the constraint violation improves as  $M$  increases. On this low dimensional distribution, increasing  $M > 1$  quickly saturates (see Appendix 5.C), resulting in diminishing returns that do not appear in the higher dimensional settings.

**Baselines comparison.** We now compare PPR to other constrained sampling methods. Figures 5.3b and 5.4, and Tables 5.2 and 5.3 show that PPR has the highest feasibility rate, the lowest constraint violation, and produces a closer approximation to the constrained distribution as measured by the Skill, RMSE, and the  $k$ -NN cross-edge rate on DATA2D

## 5.5 EXPERIMENTS



(a) We vary the number of kernel applications  $M$  ( $x_t^M$ ) and whether information is propagated through the denoiser ( $x_t^M$ ), or not ( $x_0^M$ ). Projecting in  $x_0$ -space instead of  $x_t$ -space increases constraint violation; on KS, no  $x_0$ -space sample lies in  $\mathcal{C}$ . When projecting through the denoiser in  $x_t$ -space, larger  $M$  lowers violation.

(b) PPR obtains the lowest constraint violation and highest feasibility %. The % of feasible samples is over 82% for PPR and 1% for the next best method on KS and 79% vs 31% on DATA2D.

Figure 5.3. Constraint violation for KS (top) and DATA2D (bottom) for the ablation (left) and the baselines comparison (right). Red lines show the thresholds,  $4 \times 10^{-4}$  for KS and  $4 \times 10^{-6}$  for DATA2D. Samples above are considered not in  $\mathcal{C}$ .

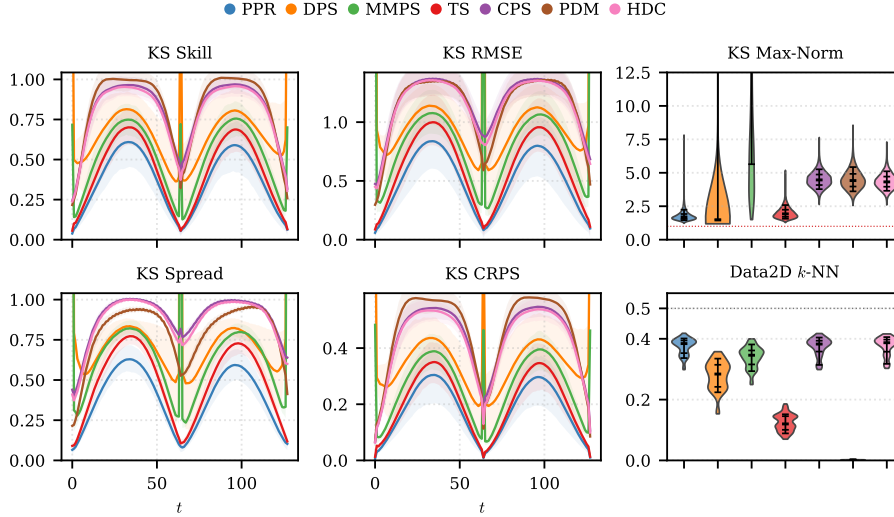


Figure 5.4. Baseline comparison on KS and DATA2D (bottom right). PPR gives the best constrained distribution in both cases. Only PPR and TS produce samples in  $\mathcal{S}$ ; all other methods yield discontinuous samples. For KS, skill, RMSE, and CRPS measure prediction accuracy; for DATA2D, the  $k$ -NN cross-edge rate measures constrained-distribution quality. Max-Norm is better when closer to 1 (dotted red line). In the KS metrics, the x-axis  $t$  denotes system time, not diffusion time.

Table 5.2. Comparison on KS. Only PPR produces feasible samples  $x \in \mathcal{C} \cap \mathcal{S}$  at a meaningful rate. The feasibility threshold was set at  $c(x) < 4 \times 10^{-4}$ . Max-Norm is better the closer it is to 1.

| Method     | Feas. (%) $\uparrow$ | RMSE $\downarrow$ | CRPS $\downarrow$ | Skill $\downarrow$ | Spread $\downarrow$ | Max-Norm    | Time (s) $\downarrow$ |
|------------|----------------------|-------------------|-------------------|--------------------|---------------------|-------------|-----------------------|
| PPR (Ours) | <b>82.7</b>          | <b>0.503</b>      | <b>0.177</b>      | <b>0.365</b>       | <b>0.385</b>        | <b>1.80</b> | 80.2                  |
| DPS        | 0.0                  | 1.089             | 0.352             | 0.666              | 0.865               | 17.11       | 25.5                  |
| MMPS       | 1.0                  | 0.830             | 0.265             | 0.519              | 0.672               | 19.28       | 307                   |
| TS         | 0.0                  | 0.614             | 0.206             | 0.427              | 0.487               | 2.01        | 398                   |
| CPS        | 0.0                  | 1.179             | 0.436             | 0.805              | 0.890               | 4.50        | 50.8                  |
| PDM        | 0.8                  | 1.137             | 0.463             | 0.831              | 0.798               | 4.48        | 296                   |
| HDC        | 0.0                  | 1.152             | 0.425             | 0.787              | 0.870               | 4.36        | 49.8                  |

and CRPS on KS. PDM, CPS, and HDC, working in  $x_0$ -space, fail to produce  $x_0 \in \mathcal{S}$  as the KS samples are discontinuous. The percentage of feasible samples is over 82% for PPR vs 1% for the next best method on KS and 79% vs 31% on DATA2D.

All methods used the same computational budget. However, many baselines do not offer an effective way to trade compute for improvements. For DPS and MMPS, the main lever is the number of diffusion steps, which quickly hits diminishing returns. CPS, PDM, and HDC also expose the number of iterations in their projection routines, but these  $x_0$ -space projections converge quickly, often to points outside  $\mathcal{S}$ , so additional iterations bring little benefit. In TS, the trust-region mechanism can terminate projection prematurely and even with a large region its normalized projection is less effective than the off-the-shelf solvers used by PPR, leading to wasted computation.

Table 5.3. Data2D comparison. The feasibility threshold is  $4 \times 10^{-6}$ . The  $k$ -NN rate is better closer to 0.5.

| Method     | Time (s)       | Feas. $\uparrow$ | $k$ -NN                             |
|------------|----------------|------------------|-------------------------------------|
| PPR (Ours) | $14.9 \pm 0.6$ | <b>79.0%</b>     | <b><math>0.375 \pm 0.026</math></b> |
| DPS        | $2.6 \pm 0.2$  | 1.0%             | $0.279 \pm 0.046$                   |
| MMPS       | $11.7 \pm 0.4$ | 3.6%             | $0.340 \pm 0.034$                   |
| Trust      | $3.8 \pm 0.2$  | 0.1%             | $0.124 \pm 0.027$                   |
| CPS        | $5.9 \pm 0.4$  | 31.6%            | $0.373 \pm 0.030$                   |
| PDM        | $10.8 \pm 0.1$ | 1.4%             | $0.002 \pm 0.000$                   |
| HDC        | $3.5 \pm 0.1$  | 30.3%            | <b><math>0.375 \pm 0.030</math></b> |

**Large-scale latent weather.** In a setting similar to KS, we treat a perfect nonlinear observation as a constraint. We use Appa [38], a score-based model that generates global atmospheric trajectories at  $0.25^\circ$  resolution and 1-hour intervals trained on ERA5 [41]. One atmospheric state has shape  $1440 \times 721 \times 71$ , which is  $\mathcal{O}(10^8)$  elements  $\approx 300$ MB. The decoder  $\mathcal{D}$ , through which we define the constraint, has 170M parameters. Appa’s denoiser generates 24-hour trajectories, from which we observe 12:00 PM through the decoder. This results in the constraint  $c(x_0) = \|\mathcal{D}(x_0^{12}) - y\|_2^2$ , where  $y$  is the value of the observation, extracted from ERA5. Figure 5.5 shows the results of the constrained sampling across 4 test dates. For each method, an ensemble of size 32 was sampled under the same computational budget and wall runtime. The computational budget is the same as [38], and was not optimized. PPR outperforms all baselines: it obtains lower constraint violation and its ensemble trajectories remain closer to the true trajectory for longer. See Appendix 5.D and 5.G for more results.



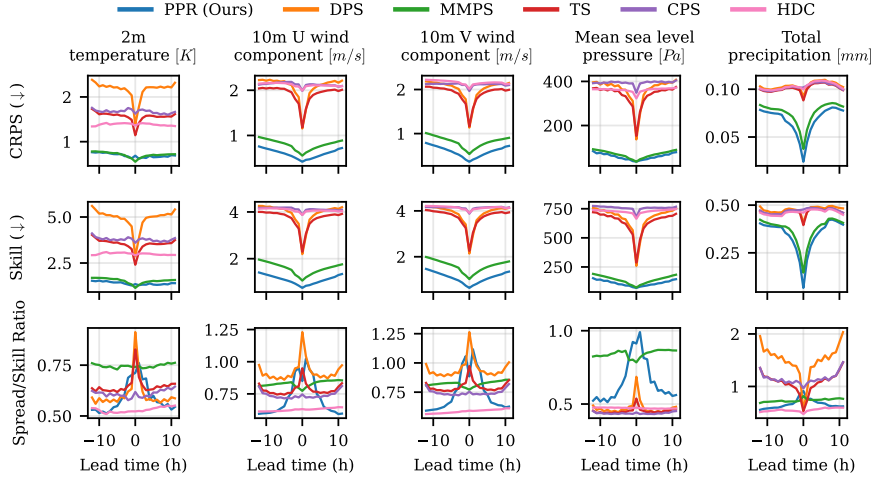


Figure 5.5. Comparison on Appa across 4 test dates. PPR outperforms all baselines on all variables. ERA5 surface variables shown. The metrics were computed with WeatherBench2 [42].

## 5.6 CONCLUSION

To address the problem of constrained sampling in score-based models, we defined a constrained forward process with its corresponding reverse process, identified a transition kernel whose unique stationary distribution is that constrained marginal, and introduced PPR as an approximation of that kernel. Over a series of experiments, we showed that PPR achieves a higher feasibility rate, lower constraint violation, and closer approximation to the constrained distribution compared to the state of the art and that each of the components of PPR, the projection through the denoiser, the renoising, and the reapplication of the kernel, is fundamental to its performance. Additionally, PPR offers a meaningful way of trading compute for sample and constrained distribution quality.

**Limitations.** We rely on approximations to sample  $p_t^c(x_0|x_t)$  (see Appendix 5.C for the error bound). PPR relies on the differentiability of the constraint function  $c(x)$  to propagate information from  $x_0$  to  $x_t$ . This dependency precludes the direct application of the framework to non-differentiable, discrete, or black-box constraints without further work.

**Future work.** A possible way of reducing the computational cost of PPR is to only project at a subset of diffusion steps. While we preliminarily found that it gives poorer constraint satisfaction and constrained distribution approximation, it is an additional lever to be tuned by the user. Further, extending beyond equality constraints to mixed and inequality constraints is a promising direction. Inequality constraints could be handled by a projection with momentum or Langevin projection that stops once inside  $C$ . Extending the framework to explicitly handle discrete constraints and domains such as inverse folding in constrained protein design also remains open for further research.

*By this art you may  
contemplate the variation  
of the 23 letters...*  
— *Anatomy of Melancholy*,  
Pt. 2, Sec. II, Mem. IV

## REFERENCES

- [1] Omer Rochman-Sharabi and Gilles Louppe. “Predict-Project-Renoise: Sampling Diffusion Models under Hard Constraints”. 2026. Submitted.
- [2] Jascha Sohl-Dickstein et al. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. 2015.
- [3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising Diffusion Probabilistic Models”. 2020.
- [4] Yang Song et al. “Score-Based Generative Modeling through Stochastic Differential Equations”. 2021.
- [5] Zihan Zhou, Xiaoxue Wang, and Tianshu Yu. “Generating Physical Dynamics under Priors”. Feb. 28, 2025. URL: <http://arxiv.org/abs/2409.00730> (visited on 04/29/2025). Pre-published.
- [6] Shervin Khalafi, Dongsheng Ding, and Alejandro Ribeiro. “Constrained Diffusion Models via Dual Training”. 2024.
- [7] Saeid Naderiparizi et al. “Constrained Generative Modeling with Manually Bridged Diffusion Models”. 2025.
- [8] Prafulla Dhariwal and Alex Nichol. “Diffusion Models Beat GANs on Image Synthesis”. 2021.
- [9] Hyungjin Chung et al. “Improving Diffusion Models for Inverse Problems using Manifold Constraints”. 2024.
- [10] Omer Rochman and Gilles Louppe. “Enforcing governing equation constraints in neural PDE solvers via training-free projections”. 2025.
- [11] Giannis Daras et al. “A Survey on Diffusion Models for Inverse Problems”. 2024.
- [12] Chieh-Hsin Lai et al. “The Principles of Diffusion Models”. 2025.
- [13] Simo Särkkä and Arno Solin. “Applied Stochastic Differential Equations”. Cambridge University Press, 2019.
- [14] Brian D.O. Anderson. “Reverse-time diffusion equation models”. In *Stochastic Processes and their Applications* 12.3 (1982), pp. 313–326.
- [15] Tero Karras et al. “Elucidating the Design Space of Diffusion-Based Generative Models”. 2022.
- [16] M. C. K. Tweedie. “Functions of a statistical variate with given means, with special reference to Laplacian distributions”. In *Mathematical Proceedings of the Cambridge Philosophical Society* (1947).
- [17] Bradley Efron. “Tweedie’s Formula and Selection Bias”. In *Journal of the American Statistical Association* (2011).
- [18] Jiaming Song, Chenlin Meng, and Stefano Ermon. “Denoising Diffusion Implicit Models”. 2022.
- [19] Dong C. Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In *Math. Program.* 45.1–3 (Aug. 1989), pp. 503–528.
- [20] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. 2017.
- [21] Hyungjin Chung et al. “Diffusion Posterior Sampling for General Noisy Inverse Problems”. 2024.
- [22] François Rozet et al. “Learning Diffusion Priors from Observations by Expectation Maximization”. 2025.



- [23] Bowen Song et al. “Solving Inverse Problems with Latent Diffusion Models via Hard Data Consistency”. 2024.
- [24] Sai Shankar Narasimhan et al. “Constrained Posterior Sampling: Time Series Generation with Hard Constraints”. 2025.
- [25] Stefano Zampini et al. “Training-Free Constrained Generation With Stable Diffusion Models”. 2025.
- [26] William Huang et al. “Constrained Diffusion with Trust Sampling”. 2024.
- [27] Jacob K Christopher, Stephen Baek, and Ferdinando Fioretto. “Constrained Synthesis with Projected Diffusion Models”. 2024.
- [28] Vivek Jayaram et al. “Linearly Constrained Diffusion Implicit Models”. 2025.
- [29] Guan-Horng Liu et al. “Mirror Diffusion Models for Constrained and Watermarked Generation”. 2024.
- [30] Vivek Jayaram et al. “Constrained Diffusion Implicit Models”. 2024.
- [31] Jacob K. Christopher et al. “Constrained Diffusion for Protein Design with Hard Structural Constraints”. 2026.
- [32] Chaoran Cheng et al. “Gradient-Free Generation for Hard-Constrained Systems”. 2025.
- [33] Xiangming Meng and Yoshiyuki Kabashima. “Diffusion Model Based Posterior Sampling for Noisy Linear Inverse Problems”. 2024.
- [34] Jiaming Song et al. “Pseudoinverse-Guided Diffusion Models for Inverse Problems”. In *International Conference on Learning Representations*. 2023.
- [35] Alexandre Galashov et al. “Learn to Guide Your Diffusion Model”. 2025.
- [36] Haotian Ye et al. “TFG: Unified Training-Free Guidance for Diffusion Models”. 2024.
- [37] Yoshiki Kuramoto. “Diffusion-Induced Chaos in Reaction Systems”. In *Progress of Theoretical Physics Supplement* 64 (Feb. 1978), pp. 346–367.
- [38] G r me Andry et al. “Appa: Bending Weather Dynamics with Latent Diffusion Models for Global Data Assimilation”. 2025.
- [39] Mark F. Schilling. “Multivariate Two-Sample Tests Based on Nearest Neighbors”. In *Journal of the American Statistical Association* 81.395 (1986), pp. 799–806.
- [40] Julia Linhart, Alexandre Gramfort, and Pedro Rodrigues. “L-C2ST: Local Diagnostics for Posterior Approximations in Simulation-Based Inference”. In *Advances in Neural Information Processing Systems* 36 (2023).
- [41] Hans Hersbach et al. “The ERA5 global reanalysis”. In *Quarterly Journal of the Royal Meteorological Society* 146.730 (2020), pp. 1999–2049.
- [42] Stephan Rasp et al. “WeatherBench 2: A benchmark for the next generation of data-driven global weather models”. 2024.
- [43] Vincent Fortin et al. “Why should ensemble spread match the RMSE of the ensemble mean?” In *Journal of Hydrometeorology* 15.4 (2014), pp. 1708–1713.
- [44] Tilmann Gneiting and Adrian E Raftery. “Strictly proper scoring rules, prediction, and estimation”. In *Journal of the American statistical Association* 102.477 (2007), pp. 359–378.
- [45] Robert J. Adler and Jonathan E. Taylor. “Gaussian Fields”. In *Random Fields and Geometry*. Springer Monographs in Mathematics. New York, NY: Springer, 2007.

- [46] G.I. Sivashinsky. “Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations”. In *Acta Astronautica* 4.11 (1977), pp. 1177–1206.
- [47] Jingyuan Liu et al. “Muon is Scalable for LLM Training”. 2025.
- [48] Sean Meyn, Richard L. Tweedie, and Peter W. Glynn. “Markov Chains and Stochastic Stability”. 2nd ed. Cambridge Mathematical Library. Cambridge University Press, 2009.

## 5.A METRICS

We use the same metrics as in Andry et al. [38], and partially reproduce their text for the definitions here. The evaluation is performed using WeatherBench2 [42] for Appa data. For the KS experiments, the metrics are the same, but with one spatial dimension removed.

**Skill.** Skill is computed as the root mean square error of the posterior mean of an ensemble compared to the ground-truth trajectory. For  $K$  ensembles each consisting of  $M$  predicted states  $\hat{x}$  of resolution  $H \times W$ , ground truth  $x$ , the skill of a single time step is computed as

$$\text{Skill} = \sqrt{\frac{1}{KHW} \sum_{k=1}^K \sum_{i=1}^H \sum_{j=1}^W \left( x_{i,j}^k - \frac{1}{M} \sum_{m=1}^M \hat{x}_{i,j}^{k,m} \right)^2}.$$

**Spread.** Ensemble spread is computed as the square root of the ensemble variance [43]:

$$\text{Spread} = \sqrt{\frac{1}{KHW} \sum_{k=1}^K \sum_{i=1}^H \sum_{j=1}^W \frac{1}{M-1} \sum_{m=1}^M \left( \hat{x}_{i,j}^{k,m} - \frac{1}{M} \sum_{n=1}^M \hat{x}_{i,j}^{k,n} \right)^2}.$$

**Spread-skill ratio.** A well-calibrated forecast should have a (corrected for ensemble size) spread-skill ratio of 1, which is a necessary but not sufficient condition. Ratios below one indicate overconfident estimations. The corrected ratio is defined as

$$\text{Ratio} = \sqrt{\frac{M+1}{M} \frac{\text{Spread}}{\text{Skill}}}.$$

**Continuous ranked probability score (CRPS).** The CRPS [44] is defined as

$$\text{CRPS} = \frac{1}{K} \sum_{k=1}^K \left( \frac{1}{M} \sum_{m=1}^M \|\hat{x}^{k,m} - x^k\|_{L_1} - \frac{1}{2M(M-1)} \sum_{m=1}^M \sum_{n=1}^M \|\hat{x}^{k,m} - \hat{x}^{k,n}\|_{L_1} \right).$$

The first term penalizes the average divergence from the ground truth, while the second term encourages spread. Therefore, the CRPS is lowest when the distribution of the ensemble matches the ground-truth distribution.

## 5.B DATA

### 5.B.1 DATA2D

We generated i.i.d. samples in  $\mathbb{R}^2$  from two parametric distributions: a uniform checkerboard distribution and a “banana” Gaussian mixture. Unless otherwise stated,

reported samples are standardized by using an empirically estimated mean  $\mu \in \mathbb{R}^2$  and standard deviation  $\sigma \in \mathbb{R}^2$ , i.e.,

$$\tilde{x} = \frac{(x - \mu)}{\sigma}.$$

The normalization statistics are estimated from Monte Carlo samples drawn from the *non-standardized* distribution.

**Checkerboard.** Fix an integer grid size  $m$  and a jitter scale  $\eta \geq 0$ . We draw  $i \sim \text{Unif}\{0, \dots, m-1\}$  and then draw  $j$  uniformly from  $\{0, \dots, m-1\}$  subject to the parity constraint  $(i + j) \bmod 2 = 0$  (equivalently, points occupy alternating squares). Given  $(i, j)$ , we draw  $u_1, u_2 \sim \mathcal{U}(0, 1)$  independently and set

$$x = \begin{bmatrix} i + u_1 \\ j + u_2 \end{bmatrix}.$$

If  $\eta > 0$ , we add isotropic Gaussian jitter,  $x \leftarrow x + \eta \xi$  with  $\xi \sim \mathcal{N}(0, I_2)$ .

**Banana Gaussian mixture model (Banana-GMM).** Let  $\{w_k\}_{k=1}^K$  be nonnegative mixture weights with  $\sum_k w_k = 1$ , and let  $\mu_k \in \mathbb{R}^2$  and  $s_k \in \mathbb{R}_+^2$  be component means and diagonal standard deviations. We first sample a component index  $k \sim \text{Categorical}(w_1, \dots, w_K)$  and then draw a Gaussian sample

$$z \sim \mathcal{N}(\mu_k, \text{diag}(s_k^2)), \quad z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}.$$

We then apply a “banana” shear in the second coordinate,

$$x_1 = z_1, \quad x_2 = z_2 + b(z_1^2 - \mathbb{E}[z_1^2]),$$

where  $b \in \mathbb{R}$  controls the curvature and the centering term  $\mathbb{E}[z_1^2]$  is computed analytically under the mixture,

$$\mathbb{E}[z_1^2] = \sum_{k=1}^K w_k ((\mu_k)_1^2 + (s_k)_1^2).$$

**GRF Constraints.** We generated a collection of 12 independent scalar constraint functions on  $\mathbb{R}^2$  as random draws from an approximate Gaussian Random Field (GRF) [45] with an RBF kernel, using random Fourier features. All constraints are defined by the same hyperparameters. Fix the number of features  $M$  (here  $M = 64$ ), lengthscale  $\ell > 0$  (here  $\ell = 0.25$  in each input dimension), kernel variance  $\sigma_f^2$  (here  $\sigma_f^2 = 1$ ), and an optional additive bias. For each constraint, we sample frequencies and phases

$$\omega_m \sim \mathcal{N}(0, \ell^{-2} I), \quad \phi_m \sim \mathcal{U}(0, 2\pi), \quad m = 1, \dots, M,$$

independently, and we sample coefficients

$$a_m \sim \mathcal{N}(0, 1), \quad m = 1, \dots, M.$$

We define the random feature map for  $x \in \mathbb{R}^2$  as

$$\varphi_m(x) = \cos(\omega_m^\top x + \phi_m),$$

and the corresponding random function

$$f(x) = \sqrt{\frac{2\sigma_f^2}{M}} \sum_{m=1}^M a_m \varphi_m(x) + \beta,$$

where the bias term is sampled as

$$\beta \sim \mathcal{N}(0, \sigma_b^2 \sigma_f^2),$$

with  $\sigma_b = 0.05$ . In the limit  $M \rightarrow \infty$ , this construction converges (in distribution) to a zero-mean GP with an RBF kernel of lengthscale  $\ell$  and marginal variance  $\sigma_f^2$ . Each constraint is obtained by smoothing  $f(x)$ ,

$$c(x) = 1 - e^{-f(x)^2}.$$

We formed the 12 constraints  $\{c_i\}_{i=1}^{12}$  by sampling  $(\{\omega_m, \phi_m, a_m\}_{m=1}^M, \beta)$  independently for each  $i$ . We used a tolerance  $\epsilon = 10^{-2}$  for the creation of the constrained distributions.

### 5.B.2 KURAMOTO-SIVASHINSKY

The 1-D Kuramoto–Sivashinsky Eq. [37, 46]

$$\partial_t u + \partial_{xx} u + \partial_{xxx} u + u \partial_x u = 0$$

is solved on the periodic domain  $x \in [0, 64]$  with a pseudospectral solver. The spatial derivatives use the FFT on the grid of size  $N = 512$  and no de-aliasing. Time stepping was done using the first-order backward-difference formula (BDF1) with fixed step  $dt = 10^{-1}$  for 1024 steps, giving  $t_{\max} = 102.4$ . During training, we used a subset of each solution, starting at step 512 and ending at step 1024, corresponding to  $t \in [51.2, 102.4]$ . Each implicit step is solved by Newton iterations. Initial data are random sums of ten cosine modes,

$$u_0(x) = \sum_{k=1}^{10} a_k \cos(2\pi\omega_k x/64 + \phi_k),$$

where amplitudes  $a_k \sim \mathcal{U}(0, 1)$ , frequencies  $\omega_k \sim \mathcal{U}(1, 5)$  and phases  $\phi_k \sim \mathcal{U}(0, 2\pi)$  are sampled independently. The dataset comprises 65,536 training and 128 validation trajectories, each generated with an independent initial condition. The test trajectories for Section 5.5 were generated with the unconditional diffusion model.

### 5.B.3 ARCHITECTURE

Both architectures use SiLU activations, sinusoidal noise-level embeddings, and EDM-style preconditioning.

**Data2D** uses a Modulated MLP: a 3-hidden-layer MLP (256 units per layer) with residual connections. The noise level  $\sigma$  and the input coordinates  $x \in \mathbb{R}^2$  and their norm are each encoded by separate sinusoidal embeddings and injected into every hidden layer via FiLM modulation.

**Kuramoto–Sivashinsky (KS)** uses a UNet with channel progression [32, 64, 128, 256]. Each scale has one FiLM-modulated residual block (3×3 convolutions) and a strided convolution (encoder) or transposed convolution (decoder) for downsampling and upsampling. A small MLP lifts the 1-channel input to the base feature width and projects back at the output. Circular padding is applied in the spatial direction; asymmetric zero-padding handles the (non-periodic) time dimension.

The noise schedule is given by

$$\sigma_t = \exp(s \logit(t(1 - 2\epsilon) + \epsilon) + \log \sigma_{\text{med}}), \quad (5.10)$$

## 5.C THEORETICAL ANALYSIS AND APPROXIMATION ERROR BOUND

$$\varepsilon = \left( \sqrt{\frac{\sigma_{\min}}{\sigma_{\max}}} \right)^{1/\alpha}, \quad \sigma_{\text{med}} = \sqrt{\sigma_{\min}\sigma_{\max}}, \quad (5.11)$$

and a constant  $\alpha_t = 1$  scale schedule. This schedule was inspired by Andry et al. [38].

Table 5.4. Architecture and noise-schedule parameters for KS and Data2D.

|                   | Data2D              | KS                 |
|-------------------|---------------------|--------------------|
| Architecture      | Mod MLP             | UNet               |
| Depth / channels  | 3 layers, 256 units | [32, 64, 128, 256] |
| Activation        | SiLU                | SiLU               |
| Time conditioning | FiLM (per layer)    | FiLM (per layer)   |
| Kernel size       | —                   | $3 \times 3$       |
| Spatial padding   | —                   | circular           |
| Noise schedule    | log logit           |                    |
| $\sigma_{\min}$   | $10^{-3}$           | $10^{-4}$          |
| $\sigma_{\max}$   | $10^2$              | $10^4$             |
| Spread $s$        | 2.0                 | 4.0                |

### 5.B.4 TRAINING

DATA2D and KS were trained for 300 epochs using the Muon optimizer [47] with no weight decay and  $k = 1$  optimizer step per update. DATA2D used a batch size of 1024, a peak learning rate of  $10^{-3}$ , no warmup, cosine decay over  $101 \times 5000$  steps, and a final learning rate of  $10^{-6}$ . KS used a batch size of 64, a peak learning rate of  $10^{-4}$ , 100 warmup steps, cosine decay over 100,000 steps, and the same final learning rate of  $10^{-6}$ . All models were trained on a single NVIDIA A5000-24GB. The KS model trained overnight while the DATA2D model trained for 4 h.

### 5.B.5 APPA

Appa was trained on ERA5 [41]. We refer the reader to Andry et al. [38] for details, including training and compute.

## 5.C THEORETICAL ANALYSIS AND APPROXIMATION ERROR BOUND

In this section we restate Proposition 1 and prove it. We also bound the error of the PPR approximation, and justify the assumptions in that bound. Both propositions assume bounded second moment for all distributions, a nondegenerate constrained set  $0 < \mathbb{P}(x_0 \in C) < 1$ , and well-defined conditional constrained posteriors  $\mathbb{P}(x_0 \in C \mid x_t) > 0$  on the relevant support.

**Proposition 2.** Let  $K_t(x'_t \mid x_t) = \int_C p(x'_t \mid x_0) p^C(x_0 \mid x_t) dx_0$  with  $p^C(x_0 \mid x_t) = \frac{p(x_0 \mid x_t) \mathbb{1}_C(x_0)}{\mathbb{P}(x_0 \in C \mid x_t)}$ . This kernel connects the unconstrained  $p_t(x_t)$  and constrained  $p_t^C(x_t)$  priors at time  $t$ . Under this kernel,  $p_t^C$  is the unique stationary distribution and  $(K_t)^M q \rightarrow p_t^C$  as  $M \rightarrow \infty$  for any initial distribution  $q$ .

*Beware of bugs in the above code; I have only proved it correct, not tried it.*

— Knuth, Notes on the van Emde Boas construction of priority dequeues, 1977

**Sketch.** The kernel  $K_t$  can be viewed as a two-step resampling procedure: starting from the current noisy variable  $x_t$ , it first samples a clean point  $x_0$  from the posterior restricted to the constraint set  $C$ , and then renoises this  $x_0$  to obtain a new  $x'_t$ . If the current  $x_t$  is already distributed according to the constrained noisy marginal  $p_t^C$ , then this posterior-resampling step recovers exactly the constrained clean distribution, and the renoising step maps it back to  $p_t^C$ . This shows stationarity. Since the renoising distribution is Gaussian, the chain can move between any regions of the state space with positive probability, which rules out multiple closed classes or periodic behavior. Therefore the stationary distribution is unique, and repeated applications of the kernel converge to  $p_t^C$ .

*Proof.* Fix  $t > 0$  and write

$$p_t^C(x_t) = \frac{1}{Z^C} \int_C p(x_t|x_0)p(x_0) dx_0, \quad Z^C = \mathbb{P}(x_0 \in C).$$

We first show that  $p_t^C$  is stationary for  $K_t$ , namely

$$\int K_t(x'_t|x_t) p_t^C(x_t) dx_t = p_t^C(x'_t).$$

By Bayes' rule,

$$p(x_0|x_t) = \frac{p(x_t|x_0)p(x_0)}{p_t(x_t)}.$$

Moreover,

$$\mathbb{P}(x_0 \in C|x_t) = \int_C p(x_0|x_t) dx_0 = \frac{\int_C p(x_t|x_0)p(x_0) dx_0}{p_t(x_t)} = \frac{Z^C p_t^C(x_t)}{p_t(x_t)}.$$

Hence, for  $x_0 \in C$ ,

$$p^C(x_0|x_t) = \frac{p(x_0|x_t) \mathbb{1}_C(x_0)}{\mathbb{P}(x_0 \in C|x_t)} = \frac{p(x_t|x_0)p(x_0)}{Z^C p_t^C(x_t)}.$$

Substituting this expression into the stationarity equation gives

$$\begin{aligned} \int K_t(x'_t|x_t) p_t^C(x_t) dx_t &= \int \int_C p(x'_t|x_0) p^C(x_0|x_t) dx_0 p_t^C(x_t) dx_t \\ &= \int \int_C p(x'_t|x_0) \frac{p(x_t|x_0)p(x_0)}{Z^C p_t^C(x_t)} dx_0 p_t^C(x_t) dx_t \\ &= \frac{1}{Z^C} \int \int_C p(x'_t|x_0) p(x_t|x_0) p(x_0) dx_0 dx_t \\ &= \frac{1}{Z^C} \int_C p(x'_t|x_0) p(x_0) \left( \int p(x_t|x_0) dx_t \right) dx_0 \\ &= \frac{1}{Z^C} \int_C p(x'_t|x_0) p(x_0) dx_0 \\ &= p_t^C(x'_t). \end{aligned}$$

Thus  $p_t^C$  is stationary under  $K_t$ .

It remains to justify uniqueness and convergence. Since  $\sigma_t > 0$ , the renoising density

$$p(x'_t | x_0) = \mathcal{N}(x'_t; \alpha_t x_0, \sigma_t^2 I)$$

is strictly positive for all  $x'_t$ . Hence  $K_t(x'_t | x_t) > 0$ , for all  $x_t, x'_t$ . Therefore  $K_t$  is Lebesgue-irreducible and aperiodic. Since  $p_t^C$  is an invariant probability measure, standard general-state-space Markov chain theory [48] implies that  $K_t$  is positive Harris recurrent, that  $p_t^C$  is the unique stationary distribution, and that for every initial distribution  $q$ ,

$$(K_t)^M q \rightarrow p_t^C \quad \text{as } M \rightarrow \infty$$

for any initial distribution  $q$  in the total variation sense. This proves the claim.  $\square$

**Proposition 3.** Assume  $x_t \mapsto d_\theta(\Pi_d(x_t))$  is  $L_t$ -Lipschitz and that  $\alpha_t L_t < 1$ . Then

$$W_2(q_t^{(M)}, p_t^C) \leq (\alpha_t L_t)^M W_2(q_t^{(0)}, \tilde{p}_t^C) + \frac{\Delta_t}{1 - \alpha_t L_t},$$

where

$$\Delta_t = \sup_{x_t} W_2(K_t^{\text{PPR}}(\cdot | x_t), K_t^*(\cdot | x_t)).$$

**Sketch.** The error is split into two effects: the finite number of PPR iterations and the fact that the PPR kernel is only an approximation of the ideal constrained kernel. The first effect is controlled by showing that the PPR update is contractive: two nearby noisy inputs are mapped to Gaussian transitions whose means remain closer by a factor at most  $\alpha_t L_t < 1$ . Hence repeated PPR steps forget the initialization geometrically fast. The second effect is the limiting bias caused by replacing the ideal kernel with the approximate one. This bias is controlled by the largest one-step discrepancy  $\Delta_t$  between the two kernels. Because the approximate kernel is contractive, these one-step errors do not accumulate without bound; instead they are amplified by at most the geometric factor  $1/(1 - \alpha_t L_t)$ . Combining the geometric mixing error with this accumulated approximation bias gives the stated bound.

*Proof.* Let

$$\rho_t := \alpha_t L_t.$$

By assumption,  $\rho_t < 1$ . We decompose

$$W_2(q_t^{(M)}, p_t^C) \leq W_2(q_t^{(M)}, \tilde{p}_t^C) + W_2(\tilde{p}_t^C, p_t^C).$$

We first bound the mixing term. The PPR kernel is

$$K_t^{\text{PPR}}(x'_t | x_t) = \mathcal{N}(x'_t; \alpha_t d_\theta(\Pi_d(x_t)), \sigma_t^2 I).$$

For any  $x_t, \tilde{x}_t$ , the two transition distributions have the same covariance. Hence

$$\begin{aligned} W_2(K_t^{\text{PPR}}(\cdot | x_t), K_t^{\text{PPR}}(\cdot | \tilde{x}_t)) &= \alpha_t \|d_\theta(\Pi_d(x_t)) - d_\theta(\Pi_d(\tilde{x}_t))\| \\ &\leq \alpha_t L_t \|x_t - \tilde{x}_t\| \\ &= \rho_t \|x_t - \tilde{x}_t\|. \end{aligned}$$

Therefore, for any probability measures  $q_1, q_2$ ,

$$W_2(K_t^{\text{PPR}} q_1, K_t^{\text{PPR}} q_2) \leq \rho_t W_2(q_1, q_2).$$

That is,  $K_t^{\text{PPR}}$  is a contraction.

Define

$$q_t^{(M)} = (K_t^{\text{PPR}})^M q_t^{(0)}$$

and let  $\tilde{p}_t^C$  be stationary for  $K_t^{\text{PPR}}$ . It exists since  $K_t^{\text{PPR}}$  is a contraction. Then repeated application gives

$$W_2(q_t^{(M)}, \tilde{p}_t^C) \leq \rho_t^M W_2(q_t^{(0)}, \tilde{p}_t^C).$$

It remains to bound the bias term. Since  $\tilde{p}_t^C$  is stationary for  $K_t^{\text{PPR}}$  and  $p_t^C$  is stationary for  $K_t^*$ ,

$$W_2(\tilde{p}_t^C, p_t^C) = W_2(K_t^{\text{PPR}} \tilde{p}_t^C, K_t^* p_t^C).$$

By the triangle inequality,

$$\begin{aligned} W_2(K_t^{\text{PPR}} \tilde{p}_t^C, K_t^* p_t^C) &\leq W_2(K_t^{\text{PPR}} \tilde{p}_t^C, K_t^{\text{PPR}} p_t^C) \\ &\quad + W_2(K_t^{\text{PPR}} p_t^C, K_t^* p_t^C). \end{aligned}$$

The first term is bounded by the contraction estimate:

$$W_2(K_t^{\text{PPR}} \tilde{p}_t^C, K_t^{\text{PPR}} p_t^C) \leq \rho_t W_2(\tilde{p}_t^C, p_t^C).$$

For the second term, coupling the same  $x_t \sim p_t^C$  and then using an optimal conditional coupling of the two kernels gives

$$\begin{aligned} W_2(K_t^{\text{PPR}} p_t^C, K_t^* p_t^C) &\leq \sup_{x_t} W_2(K_t^{\text{PPR}}(\cdot|x_t), K_t^*(\cdot|x_t)) \\ &= \Delta_t. \end{aligned}$$

Thus

$$W_2(\tilde{p}_t^C, p_t^C) \leq \rho_t W_2(\tilde{p}_t^C, p_t^C) + \Delta_t.$$

Since  $\rho_t < 1$ , rearranging yields

$$W_2(\tilde{p}_t^C, p_t^C) \leq \frac{\Delta_t}{1 - \rho_t} = \frac{\Delta_t}{1 - \alpha_t L_t}.$$

Combining the mixing and bias bounds,

$$\begin{aligned} W_2(q_t^{(M)}, p_t^C) &\leq W_2(q_t^{(M)}, \tilde{p}_t^C) + W_2(\tilde{p}_t^C, p_t^C) \\ &\leq (\alpha_t L_t)^M W_2(q_t^{(0)}, \tilde{p}_t^C) + \frac{\Delta_t}{1 - \alpha_t L_t}. \end{aligned}$$

□

Figure 5.6 shows how the Lipschitz constant  $L_t$  evolves over diffusion time, measured by taking the maximum constant from all the constraints and sampling trajectories of DATA2D. The figure supports the assumptions that  $x_t \mapsto d_\theta(\Pi_d(x_t))$  is  $L_t$ -Lipschitz and that  $\alpha_t L_t < 1$ . Here  $\alpha_t = 1$ .

## 5.D DETAILED RESULTS

In this section we provide the detailed numerical results of the experiments from Section 5.5 in the main text. Table 5.5 extends Table 5.2 from the main text with the number of calls to the denoiser and includes other simpler constraints (Appendix 5.F) in the statistics. Tables 5.6, 5.7, 5.8, 5.9, and 5.10 give the average over time of the values in Figure 5.5.



## 5.D DETAILED RESULTS

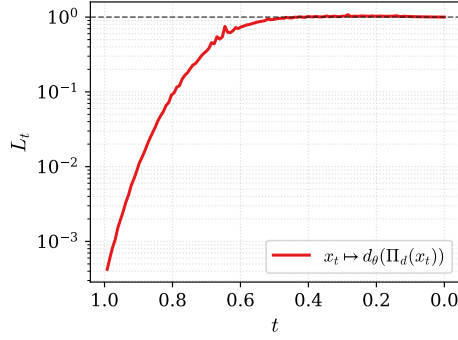


Figure 5.6. Lipschitz constant of  $x_t \mapsto d_\theta(\Pi_d(x_t))$ . It was measured numerically by evaluating the Lipschitz constant for each of the 12 constraints across 1024 samples at every timestep, then taking the maximum value.

Table 5.5. KS results. T: diffusion steps. fwd is the number of network call. bwd is the number of gradient calls (adds one to fwd too). The feasibility shows the % of samples under a threshold (0 is worse, 100 is the best). Max-Norm is better close to 1. Note that the feasibility is averaged over different constraints. For the most challenging constraints, the feasibility of PPR is over 84% compared to 1% for the next best method (main text).

| Method     | T    | fwd   | bwd   | Time (s)   | Feas. ↑      | Skill ↓              | RMSE ↓               | CRPS ↓               | Spread ↓             | MaxNorm            |
|------------|------|-------|-------|------------|--------------|----------------------|----------------------|----------------------|----------------------|--------------------|
| PPR (Ours) | 64   | 1216  | 1024  | 84.8 ± 8.1 | <b>86.9%</b> | <b>0.648 ± 0.212</b> | <b>0.891 ± 0.285</b> | <b>0.351 ± 0.130</b> | <b>0.644 ± 0.189</b> | <b>1.65 ± 0.17</b> |
| DPS        | 1024 | 2048  | 1024  | 25.6 ± 0.3 | 11.0%        | 0.817 ± 0.153        | 1.240 ± 0.165        | 0.450 ± 0.099        | 0.956 ± 0.129        | 16.64 ± 22.82      |
| MMPS       | 1024 | 5120  | 20480 | 306 ± 3    | 42.1%        | 0.735 ± 0.161        | 1.069 ± 0.179        | 0.399 ± 0.101        | 0.801 ± 0.101        | 9.08 ± 10.30       |
| Trust      | 192  | 24960 | 12288 | 405 ± 11   | 0.0%         | 0.667 ± 0.181        | 0.929 ± 0.233        | 0.358 ± 0.114        | 0.682 ± 0.144        | 1.71 ± 0.25        |
| CPS        | 1024 | 1024  | 0     | 52.6 ± 3.1 | 36.0%        | 0.873 ± 0.062        | 1.242 ± 0.055        | 0.483 ± 0.042        | 0.911 ± 0.022        | 2.82 ± 1.22        |
| PDM        | 384  | 24576 | 0     | 296 ± 3    | 56.9%        | 0.897 ± 0.064        | 1.211 ± 0.066        | 0.508 ± 0.043        | 0.837 ± 0.045        | 2.97 ± 1.10        |
| HDC        | 1024 | 1024  | 0     | 49.5 ± 0.9 | 36.3%        | 0.865 ± 0.068        | 1.229 ± 0.063        | 0.478 ± 0.046        | 0.902 ± 0.027        | 2.76 ± 1.16        |

Table 5.6. Appa results for 2m temperature [K]. Time is the wallclock to produce 32 samples sequentially. All values are mean ± std across the 24 timesteps.

| Method     | Time   | Skill ↓              | CRPS ↓               | Spread ↓             | Spread/Skill         |
|------------|--------|----------------------|----------------------|----------------------|----------------------|
| PPR (Ours) | 2h 01m | <b>1.399 ± 0.080</b> | <b>0.693 ± 0.045</b> | <b>0.807 ± 0.069</b> | 0.589 ± 0.072        |
| MMPS       | 2h 05m | 1.536 ± 0.134        | 0.708 ± 0.055        | 1.129 ± 0.102        | <b>0.746 ± 0.009</b> |
| TS         | 1h 53m | 3.525 ± 0.295        | 1.564 ± 0.105        | 2.238 ± 0.109        | 0.648 ± 0.041        |
| CPS        | 1h 26m | 3.786 ± 0.104        | 1.675 ± 0.040        | 2.274 ± 0.077        | 0.610 ± 0.017        |
| HDC        | 1h 26m | 2.996 ± 0.053        | 1.385 ± 0.024        | 1.562 ± 0.019        | 0.530 ± 0.012        |
| DPS        | 2h 15m | 4.918 ± 0.542        | 2.165 ± 0.196        | 2.892 ± 0.126        | 0.605 ± 0.072        |

Table 5.7. Appa results for 10m U-wind component [m/s]. Time is the wallclock to produce 32 samples sequentially. All values are mean ± std across the 24 timesteps.

| Method     | Time   | Skill ↓              | CRPS ↓               | Spread ↓             | Spread/Skill         |
|------------|--------|----------------------|----------------------|----------------------|----------------------|
| PPR (Ours) | 2h 01m | <b>1.104 ± 0.181</b> | <b>0.599 ± 0.095</b> | <b>0.755 ± 0.053</b> | 0.715 ± 0.136        |
| MMPS       | 2h 05m | 1.581 ± 0.236        | 0.796 ± 0.108        | 1.293 ± 0.203        | 0.829 ± 0.021        |
| TS         | 1h 53m | 3.718 ± 0.374        | 1.923 ± 0.194        | 2.858 ± 0.222        | 0.784 ± 0.044        |
| CPS        | 1h 26m | 4.122 ± 0.067        | 2.129 ± 0.041        | 3.002 ± 0.108        | 0.740 ± 0.026        |
| HDC        | 1h 26m | 4.075 ± 0.065        | 2.117 ± 0.039        | 2.518 ± 0.033        | 0.628 ± 0.011        |
| DPS        | 2h 15m | 3.923 ± 0.461        | 2.072 ± 0.244        | 3.537 ± 0.281        | <b>0.924 ± 0.077</b> |

Table 5.8. Appa results for 10m V-wind component [m/s]. Time is the wallclock to produce 32 samples sequentially. All values are mean  $\pm$  std across the 24 timesteps.

| Method     | Time   | Skill $\downarrow$                  | CRPS $\downarrow$                   | Spread $\downarrow$                 | Spread/Skill                        |
|------------|--------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| PPR (Ours) | 2h 01m | <b>1.098 <math>\pm</math> 0.224</b> | <b>0.594 <math>\pm</math> 0.117</b> | <b>0.770 <math>\pm</math> 0.061</b> | 0.741 $\pm$ 0.160                   |
| MMPS       | 2h 05m | 1.599 $\pm$ 0.250                   | 0.815 $\pm$ 0.122                   | 1.297 $\pm$ 0.219                   | 0.822 $\pm$ 0.021                   |
| TS         | 1h 53m | 3.722 $\pm$ 0.391                   | 1.907 $\pm$ 0.196                   | 2.910 $\pm$ 0.230                   | 0.798 $\pm$ 0.046                   |
| CPS        | 1h 26m | 4.137 $\pm$ 0.058                   | 2.114 $\pm$ 0.033                   | 3.042 $\pm$ 0.110                   | 0.747 $\pm$ 0.025                   |
| HDC        | 1h 26m | 4.140 $\pm$ 0.066                   | 2.159 $\pm$ 0.045                   | 2.389 $\pm$ 0.055                   | 0.586 $\pm$ 0.015                   |
| DPS        | 2h 15m | 3.854 $\pm$ 0.460                   | 2.006 $\pm$ 0.233                   | 3.570 $\pm$ 0.286                   | <b>0.949 <math>\pm</math> 0.079</b> |

Table 5.9. Appa results for Mean sea level pressure [Pa]. Time is the wallclock to produce 32 samples sequentially. All values are mean  $\pm$  std across the 24 timesteps.

| Method     | Time   | Skill $\downarrow$                     | CRPS $\downarrow$                     | Spread $\downarrow$                  | Spread/Skill                        |
|------------|--------|----------------------------------------|---------------------------------------|--------------------------------------|-------------------------------------|
| PPR (Ours) | 2h 01m | <b>111.186 <math>\pm</math> 23.199</b> | <b>59.473 <math>\pm</math> 13.391</b> | <b>68.678 <math>\pm</math> 5.978</b> | 0.653 $\pm$ 0.141                   |
| MMPS       | 2h 05m | 141.364 $\pm$ 32.115                   | 69.818 $\pm$ 15.414                   | 117.677 $\pm$ 27.700                 | <b>0.843 <math>\pm</math> 0.024</b> |
| TS         | 1h 53m | 636.298 $\pm$ 92.946                   | 337.614 $\pm$ 47.335                  | 281.827 $\pm$ 36.586                 | 0.452 $\pm$ 0.020                   |
| CPS        | 1h 26m | 756.719 $\pm$ 17.679                   | 393.572 $\pm$ 10.043                  | 325.945 $\pm$ 8.827                  | 0.437 $\pm$ 0.006                   |
| HDC        | 1h 26m | 727.634 $\pm$ 16.358                   | 361.569 $\pm$ 9.251                   | 338.532 $\pm$ 7.344                  | 0.472 $\pm$ 0.003                   |
| DPS        | 2h 15m | 667.376 $\pm$ 108.510                  | 361.394 $\pm$ 57.688                  | 306.990 $\pm$ 38.747                 | 0.474 $\pm$ 0.048                   |

Table 5.10. Appa results for Total precipitation [mm]. Time is the wallclock to produce 32 samples sequentially. All values are mean  $\pm$  std across the 24 timesteps.

| Method     | Time   | Skill $\downarrow$                  | CRPS $\downarrow$                   | Spread $\downarrow$                 | Spread/Skill                        |
|------------|--------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| PPR (Ours) | 2h 01m | <b>0.345 <math>\pm</math> 0.084</b> | <b>0.068 <math>\pm</math> 0.014</b> | <b>0.220 <math>\pm</math> 0.042</b> | 0.666 $\pm$ 0.086                   |
| MMPS       | 2h 05m | 0.371 $\pm$ 0.066                   | 0.075 $\pm$ 0.011                   | 0.269 $\pm$ 0.047                   | 0.737 $\pm$ 0.029                   |
| TS         | 1h 53m | 0.464 $\pm$ 0.018                   | 0.101 $\pm$ 0.004                   | 0.495 $\pm$ 0.092                   | <b>1.079 <math>\pm</math> 0.190</b> |
| CPS        | 1h 26m | 0.470 $\pm$ 0.012                   | 0.102 $\pm$ 0.003                   | 0.532 $\pm$ 0.047                   | 1.149 $\pm$ 0.104                   |
| HDC        | 1h 26m | 0.462 $\pm$ 0.012                   | 0.103 $\pm$ 0.003                   | 0.251 $\pm$ 0.017                   | 0.552 $\pm$ 0.031                   |
| DPS        | 2h 15m | 0.476 $\pm$ 0.018                   | 0.104 $\pm$ 0.004                   | 0.708 $\pm$ 0.150                   | 1.503 $\pm$ 0.299                   |

## 5.E ADDITIONAL ABLATIONS

This appendix reports three additional experiments that complement the main-text ablation and baselines comparison: the number of projection steps  $\Pi_d(x_t) \equiv \text{PROJECT} = \arg \min_x c(d_\theta(x))$  (Figure 5.7 and Table 5.11), the optimizer used for this inner projection problem (Figure 5.8), and a DPS predictor-corrector variant (DPS-PC) (Figure 5.9).

Table 5.11. Effect of the number of projection steps on PPR performance. Columns as in the main text.

| Projection steps | Data2D               |                                     |                       | KS                   |                                     |                                     |                                     |                                   |                       |
|------------------|----------------------|-------------------------------------|-----------------------|----------------------|-------------------------------------|-------------------------------------|-------------------------------------|-----------------------------------|-----------------------|
|                  | Feas. (%) $\uparrow$ | KNN                                 | Time (s) $\downarrow$ | Feas. (%) $\uparrow$ | RMSE $\downarrow$                   | CRPS $\downarrow$                   | Skill $\downarrow$                  | Max-Norm                          | Time (s) $\downarrow$ |
| 2                | 62.0                 | 0.336 $\pm$ 0.036                   | 9.4 $\pm$ 0.4         | 33.3                 | 1.140 $\pm$ 0.171                   | 0.445 $\pm$ 0.093                   | 0.825 $\pm$ 0.133                   | 1.91 $\pm$ 0.44                   | 39.1 $\pm$ 2.1        |
| 4                | 76.0                 | 0.371 $\pm$ 0.027                   | 10.4 $\pm$ 0.5        | 73.2                 | 1.069 $\pm$ 0.218                   | 0.403 $\pm$ 0.111                   | 0.770 $\pm$ 0.167                   | 1.79 $\pm$ 0.31                   | 53.0 $\pm$ 2.2        |
| 8                | 78.5                 | <b>0.374 <math>\pm</math> 0.027</b> | 14.1 $\pm$ 0.8        | 87.2                 | 1.012 $\pm$ 0.605                   | 0.361 $\pm$ 0.134                   | 0.711 $\pm$ 0.229                   | 1.65 $\pm$ 0.29                   | 83.7 $\pm$ 6.6        |
| 16               | 78.2                 | 0.373 $\pm$ 0.027                   | 21.3 $\pm$ 1.2        | 94.0                 | 0.925 $\pm$ 0.340                   | 0.337 $\pm$ 0.150                   | 0.666 $\pm$ 0.256                   | <b>1.60 <math>\pm</math> 0.21</b> | 150 $\pm$ 25          |
| 32               | 78.0                 | 0.370 $\pm$ 0.029                   | 35.3 $\pm$ 2.0        | <b>95.9</b>          | <b>0.904 <math>\pm</math> 0.440</b> | <b>0.330 <math>\pm</math> 0.162</b> | <b>0.652 <math>\pm</math> 0.285</b> | 1.61 $\pm$ 0.41                   | 305 $\pm$ 97          |

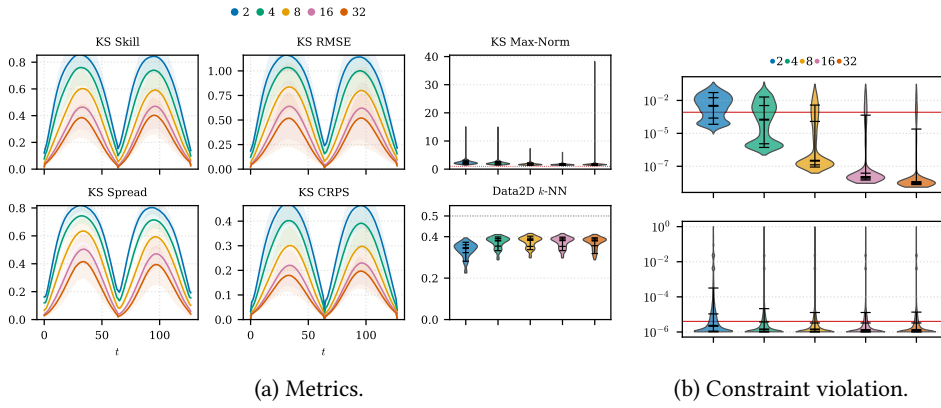


Figure 5.7. Projection-step ablation on KS and DATA2D. The labels 2, 4, 8, 16, 32 denote the number of projection iterations used inside  $\Pi_d$ . More projection iterations improve the local projection solve and reduce violation in both settings. The gains are clearest on KS; on DATA2D, the  $k$ -NN cross-edge rate improves quickly and then starts to saturate, consistent with the lower-dimensional projection problem reaching a good local minimum after relatively few iterations.

## 5.E ADDITIONAL ABLATIONS

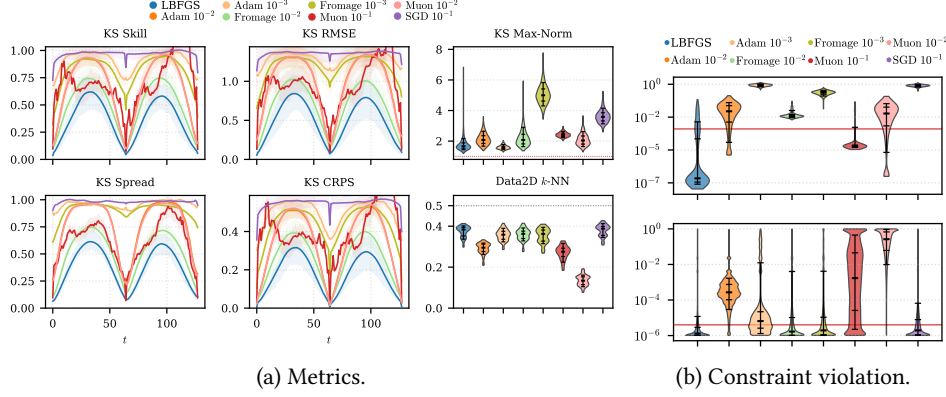


Figure 5.8. Optimizer ablation for the denoiser-space projection problem. We compare L-BFGS, Adam, Fromage, Muon, and SGD with representative learning rates; divergent learning rates have been omitted. L-BFGS is the strongest and most stable optimizer across metrics and violation, while the first-order methods are more sensitive to step size and generally give worse constrained distributions or less regular KS trajectories.

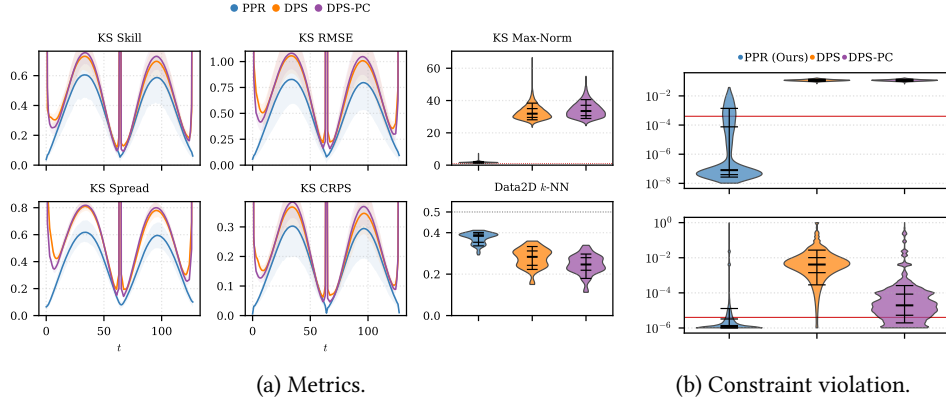


Figure 5.9. DPS-PC ablation. DPS-PC adds DPS-style correction steps in a predictor-corrector scheme. Although this can improve some feasibility-related behavior relative to DPS, it does not close the gap to PPR. The KS panels show that DPS-PC does not recover the accuracy and regularity obtained by projecting through the denoiser and renoising, and PPR remains the most reliable method for low-violation samples that preserve the data distribution.

## 5.F ADDITIONAL KS CONSTRAINTS

This section repeats the KS experiments on two additional, easier constraints. The main text uses the multi-time sine constraint

$$c(u_0) = \sum_i \|\sin(u_0^i) - \sin(u^i)\|^2, \quad i \in \{0, 64, 127\},$$

which constrains nonlinear observations at three system times. Here we also consider the single-time sine constraint  $c(u_0) = \|\sin(u_0^0) - \sin(u^0)\|^2$  and the linear observation  $c(u_0) = \|u_0^0 - u^0\|^2$ . These alternatives are substantially less challenging, especially the linear observation. Consequently, the baselines can appear more competitive on the metrics for these constraints. The relevant distinction is whether a method also maintains feasibility and continuity; PPR remains the most reliable method under that stricter criterion.

Each figure below aggregates the two additional constraints, with the single-time sine observation on the left and the linear observation on the right. Within each block, the figure reports constraint violation, Max-Norm, spread, skill, RMSE, and CRPS. In all metric panels, the x-axis is the dynamical system time, not diffusion time.

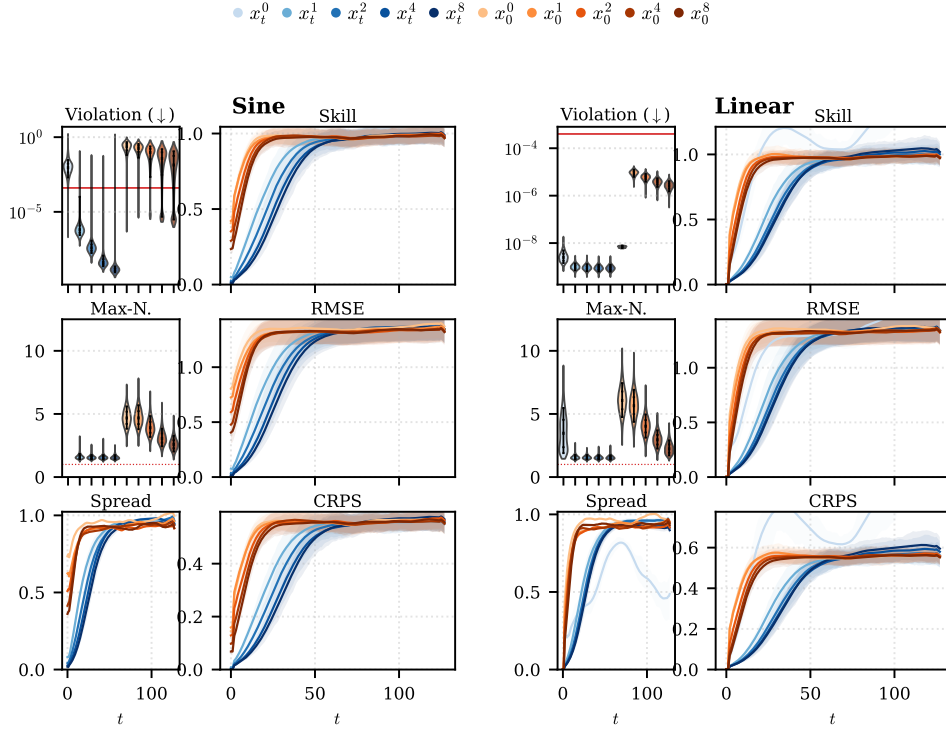


Figure 5.10. KS PPR ablation on additional constraints. Despite the easier single-time sine and linear observations, the same conclusion holds: projecting through the denoiser and renoising are necessary to preserve regular trajectories while reducing constraint violation. These easier constraints are less discriminating than the main-text multi-time sine constraint, but they still show the same failure mode for projections that do not preserve the data manifold.

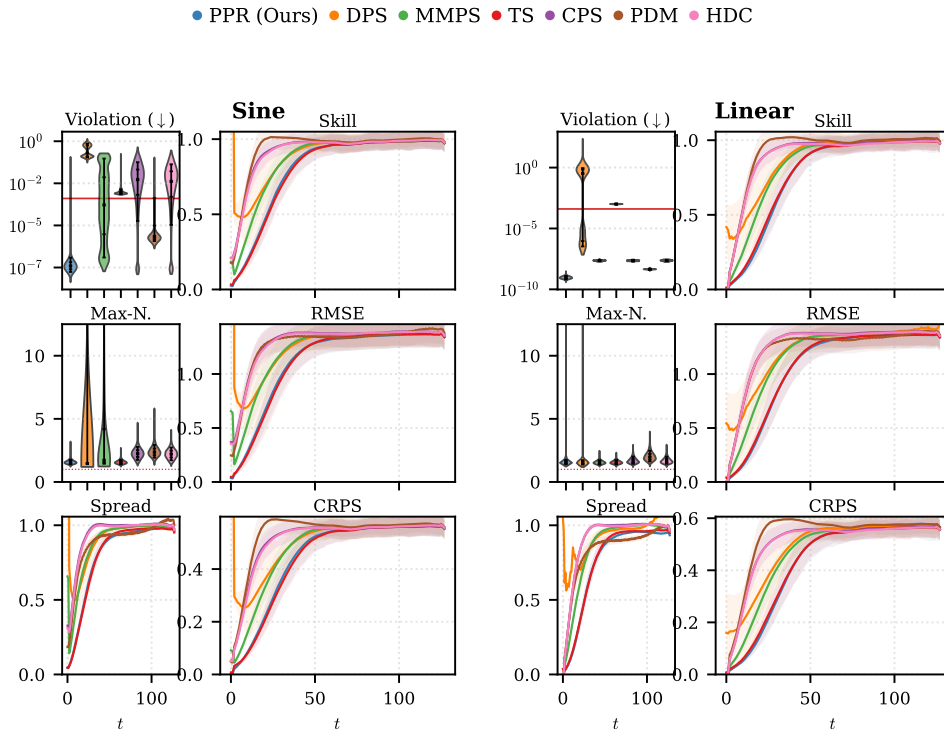


Figure 5.11. KS baseline comparison on additional constraints. Because the single-time sine and linear observations are easier, several baselines become more competitive on the trajectory proxy metrics. However, PPR remains the most consistent method when metrics are considered together with the constraint violation.

## 5.F ADDITIONAL KS CONSTRAINTS

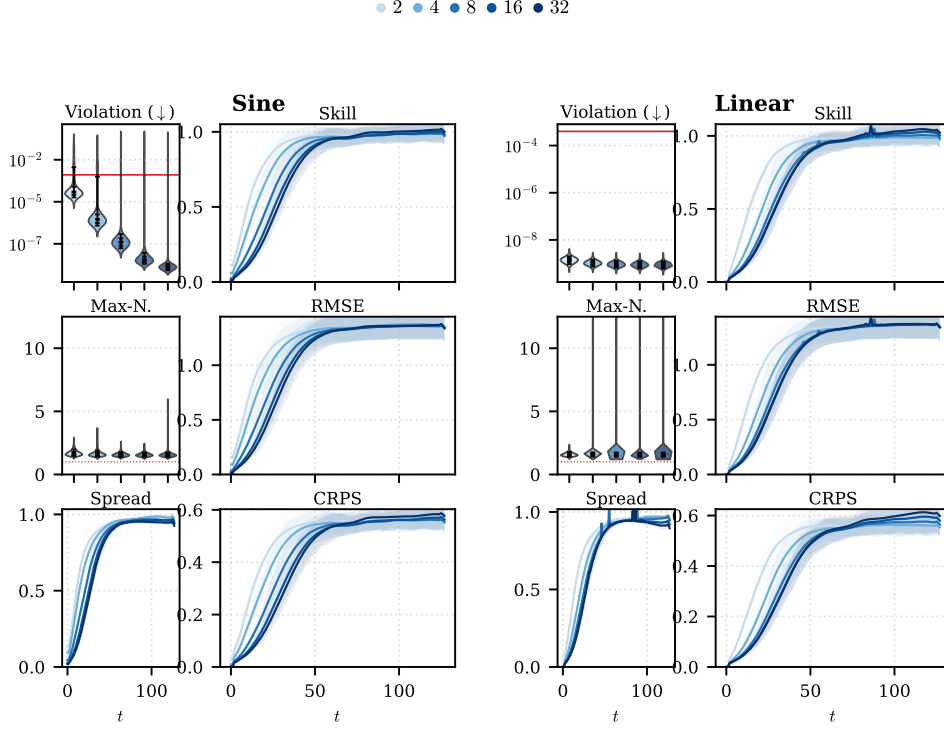


Figure 5.12. KS projection-step ablation on additional constraints. Increasing the number of projection iterations improves the local solution of  $\Pi_d$  and generally lowers violation.

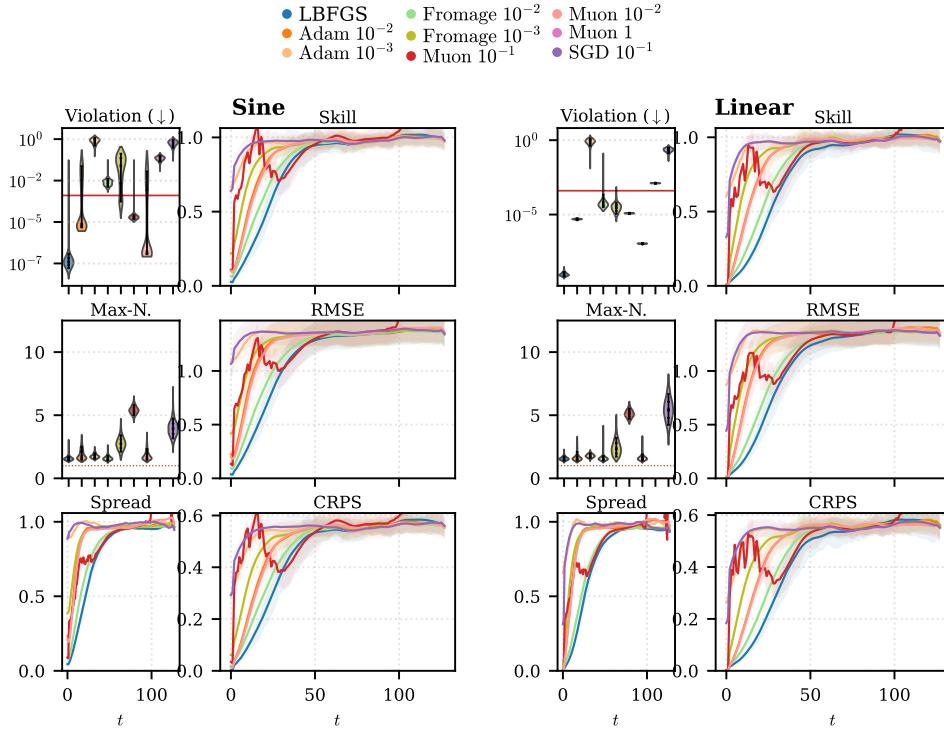


Figure 5.13. KS optimizer ablation on additional constraints. L-BFGS remains the most stable option across constraints, while first-order optimizers are more sensitive to their learning rates and less reliable in maintaining both feasibility and trajectory regularity.



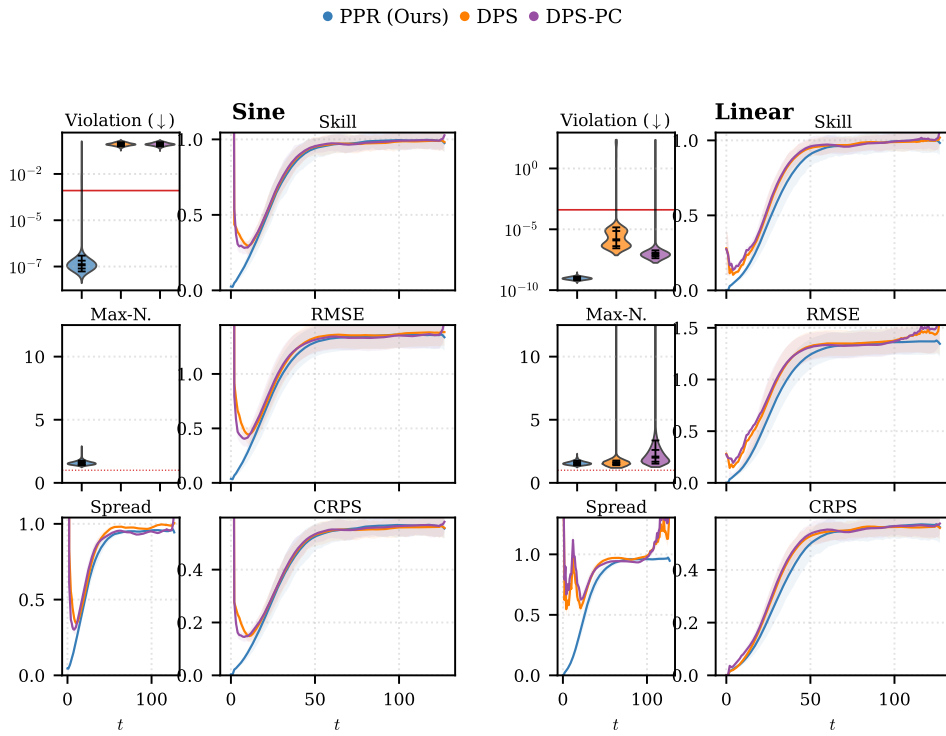


Figure 5.14. KS DPS-PC ablation on additional constraints. DPS-PC improves over DPS in some easier cases, but this does not close the gap to PPR.

## 5.G EXAMPLE SAMPLES FROM ALL EXPERIMENTS

We provide below some representative samples from each of the experiments. For KS and WEATHER, we simply picked a random ensemble member from each method to plot. Higher resolution versions are available in the supplementary material.

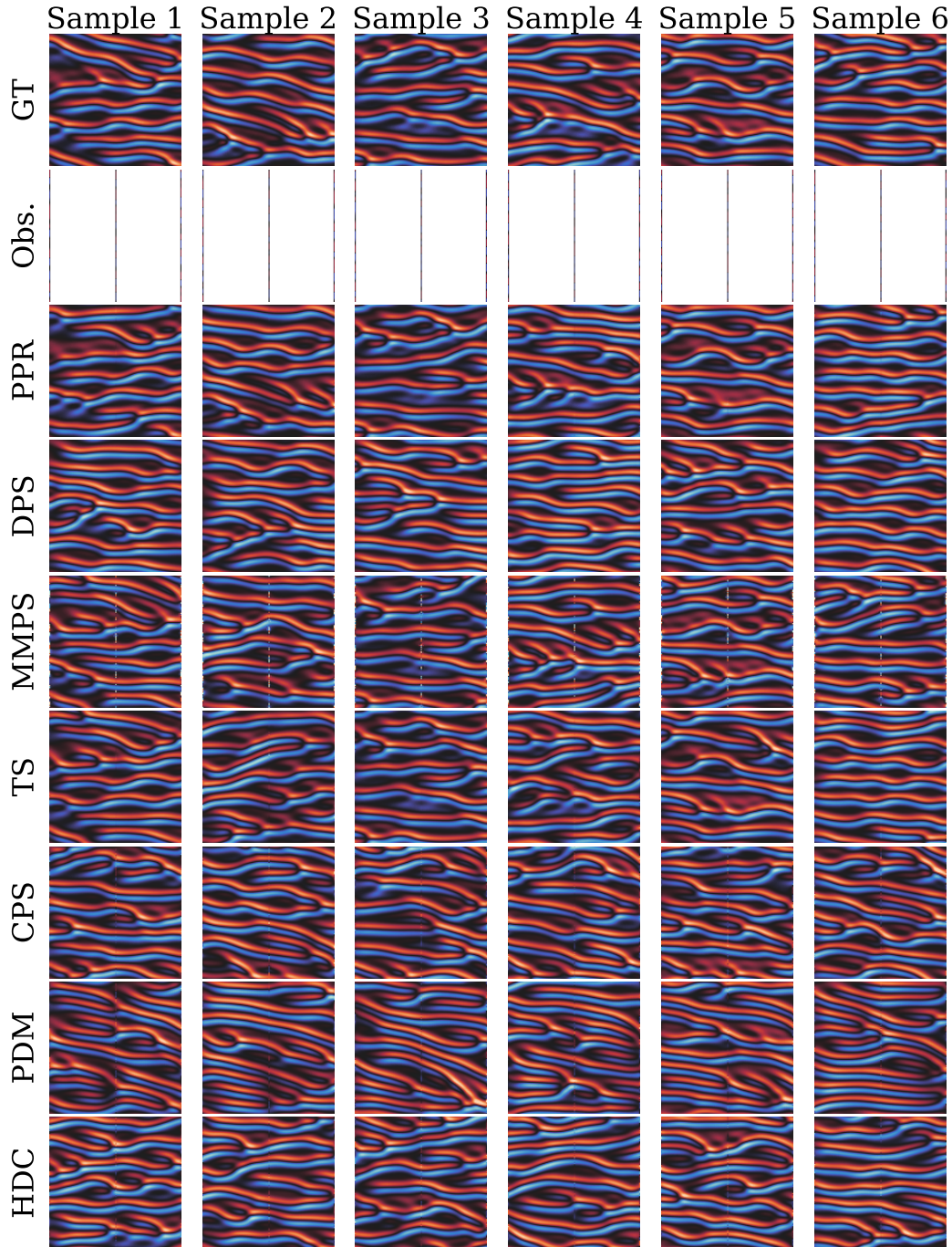


Figure 5.15. Samples from the sine observation at 3 different times, corresponding to the experiment in Section 5.5. For each sample from the ground truth set, we show one member of the ensemble produced by each method. The  $y$ -axis is the spatial domain, while the  $x$ -axis is the temporal domain.

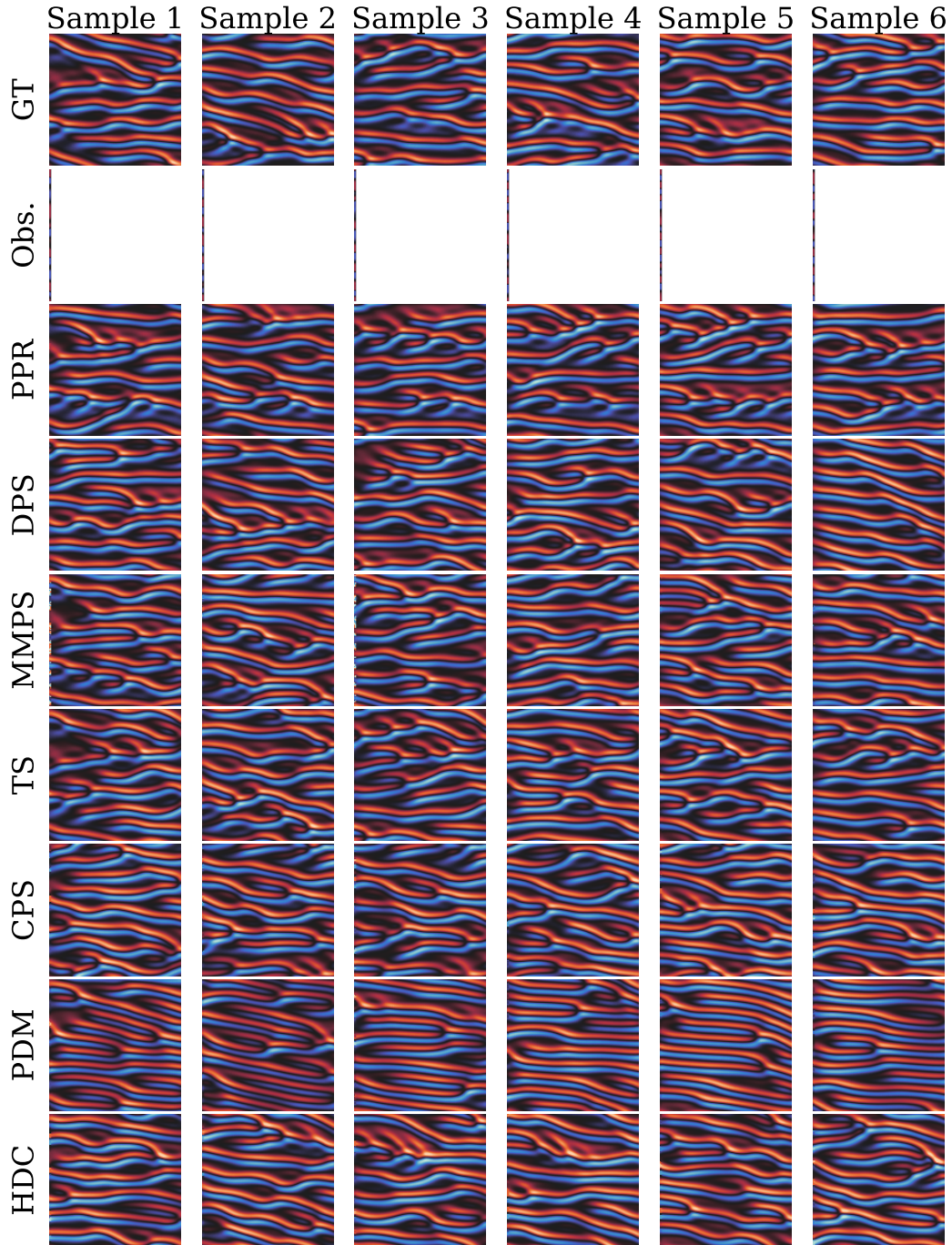


Figure 5.16. Samples from the sine observation at  $t = 0$ . For each sample from the ground truth set, we show one member of the ensemble produced by each method. The  $y$ -axis is the spatial domain, while the  $x$ -axis is the temporal domain.



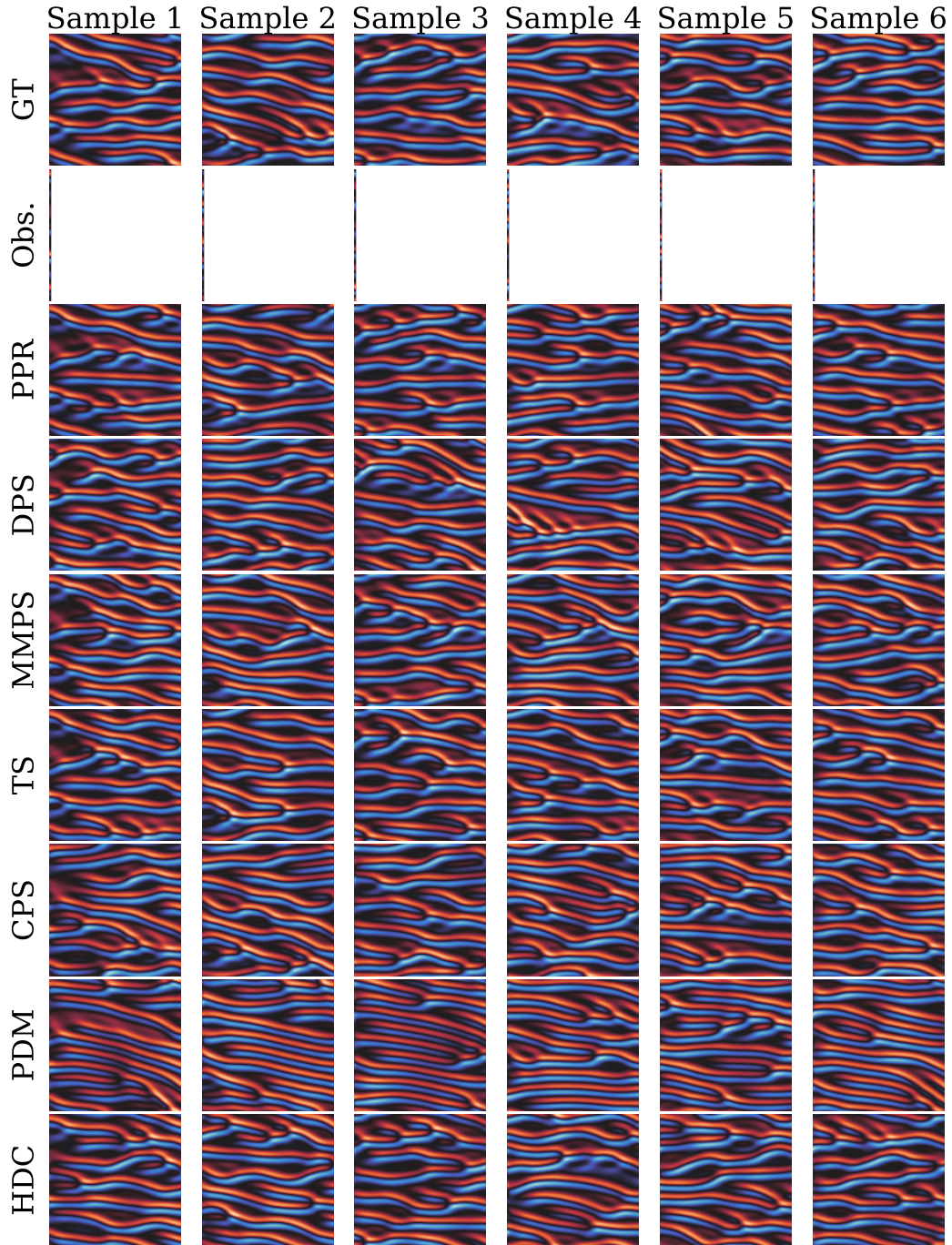


Figure 5.17. Samples from the linear observation at  $t = 0$ . For each sample from the ground truth set, we show one member of the ensemble produced by each method. The  $y$ -axis is the spatial domain, while the  $x$ -axis is the temporal domain.

## 5.G EXAMPLE SAMPLES FROM ALL EXPERIMENTS

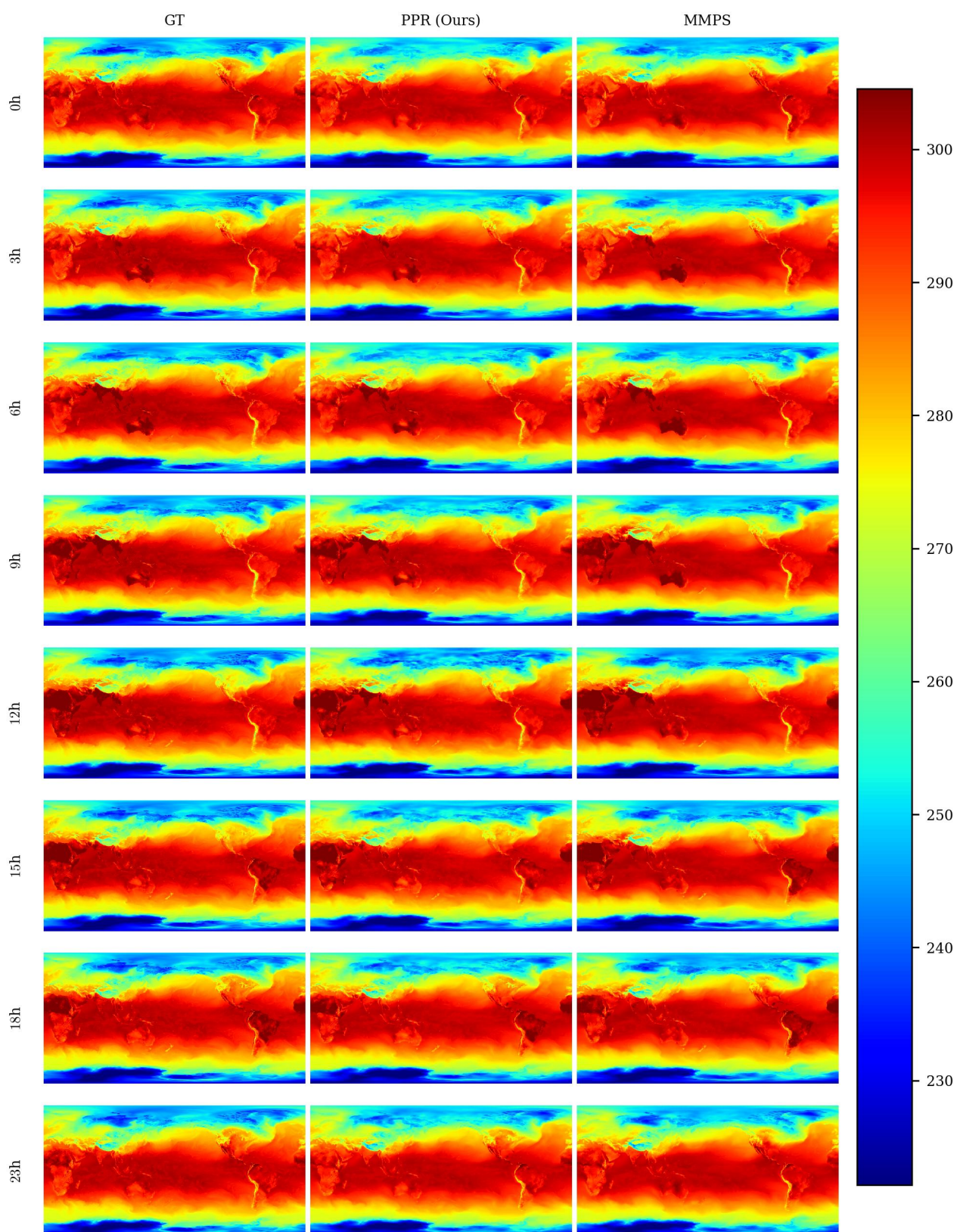


Figure 5.18. WEATHER sample showing the 2m temperature (K) for the ground truth, PPR, and MMPS. The y-axis is the hour in 24 h format.



## 5.G EXAMPLE SAMPLES FROM ALL EXPERIMENTS

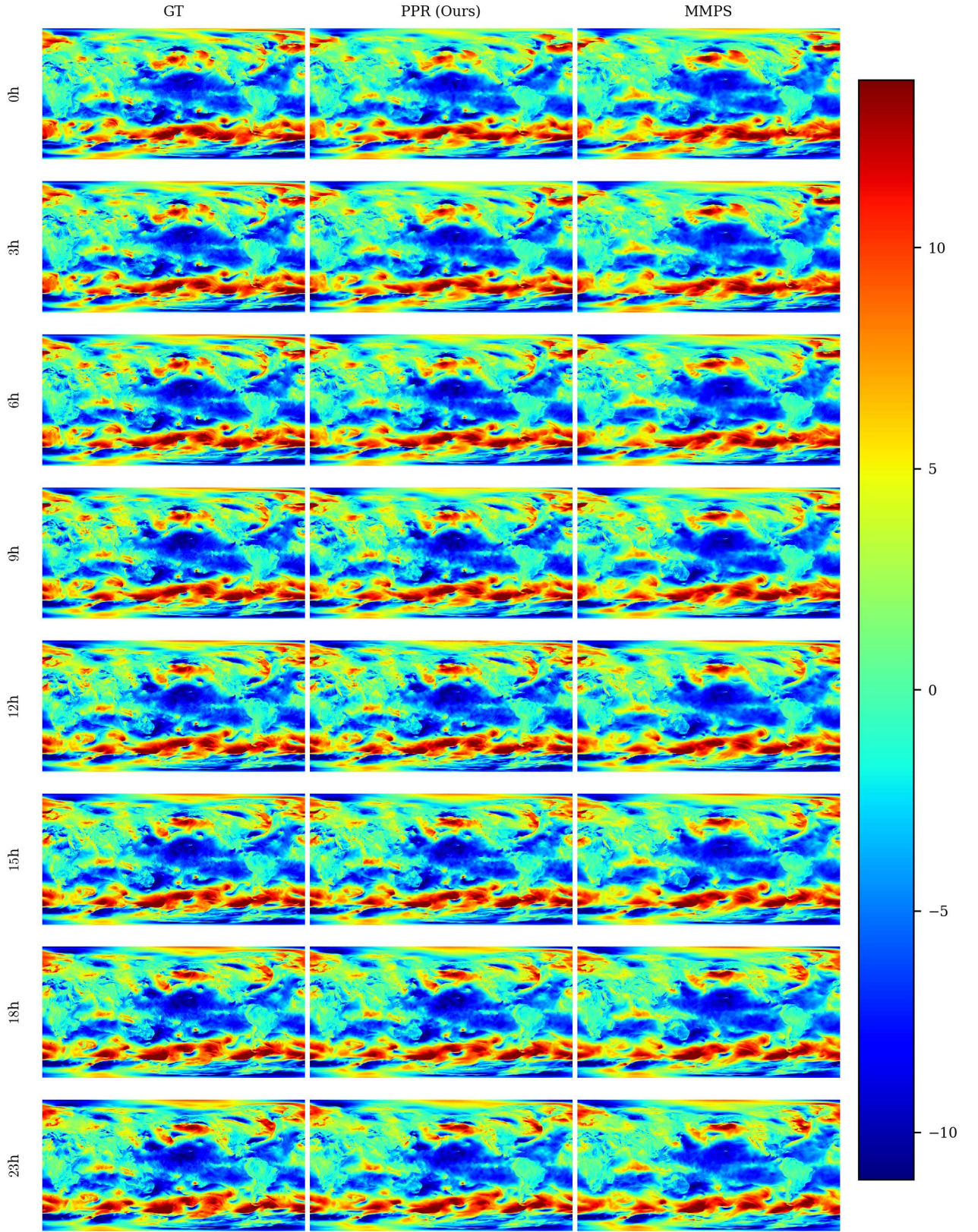


Figure 5.19. WEATHER sample showing the 10m zonal wind component  $u$  ( $\frac{m}{s}$ ) for the ground truth, PPR, and MMPS. The y-axis is the hour in 24 h format.



## 5.G EXAMPLE SAMPLES FROM ALL EXPERIMENTS

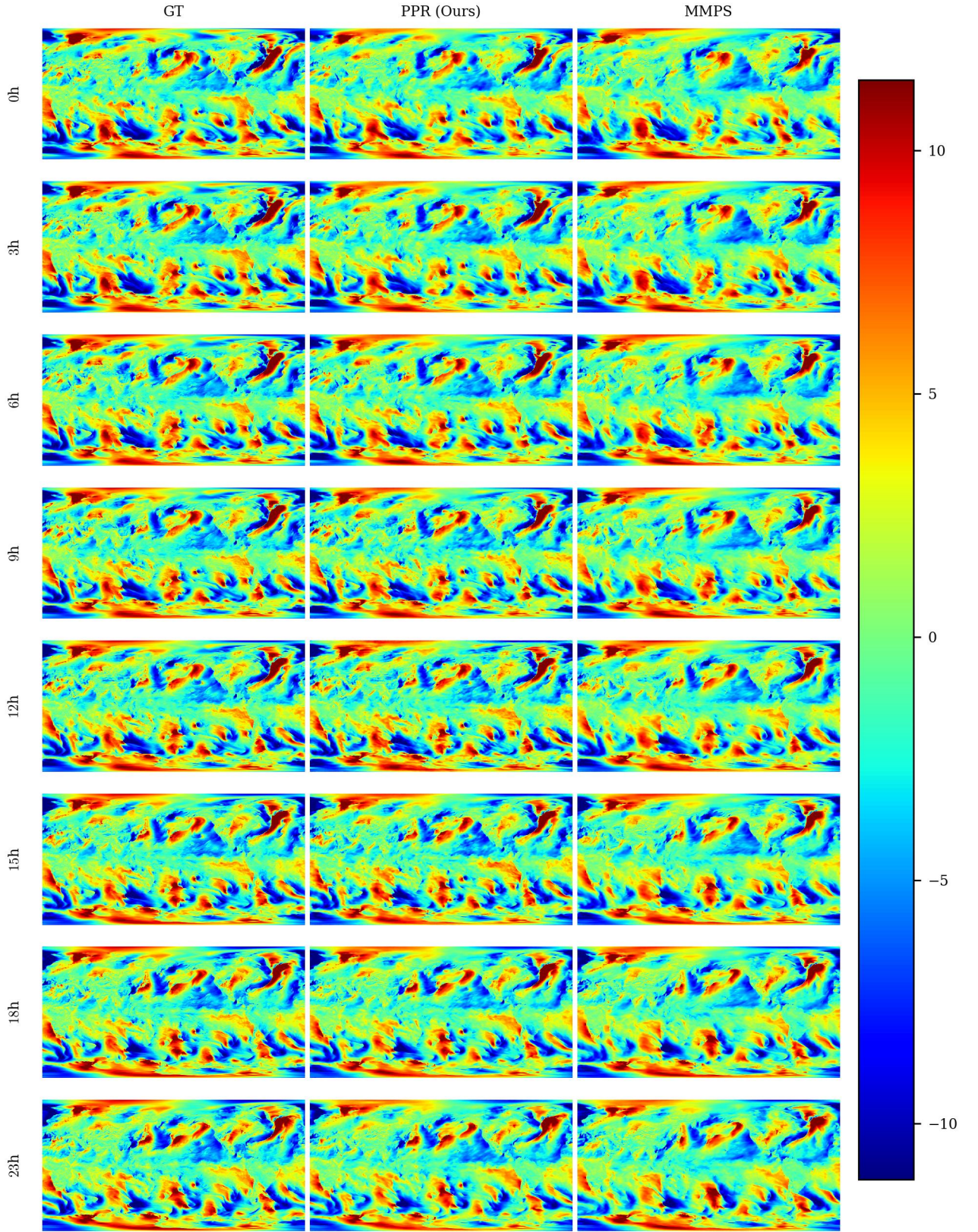


Figure 5.20. WEATHER sample showing the 10m meridional wind component  $v$  ( $\frac{m}{s}$ ) for the ground truth, PPR, and MMPS. The y-axis is the hour in 24 h format.



## 5.G EXAMPLE SAMPLES FROM ALL EXPERIMENTS

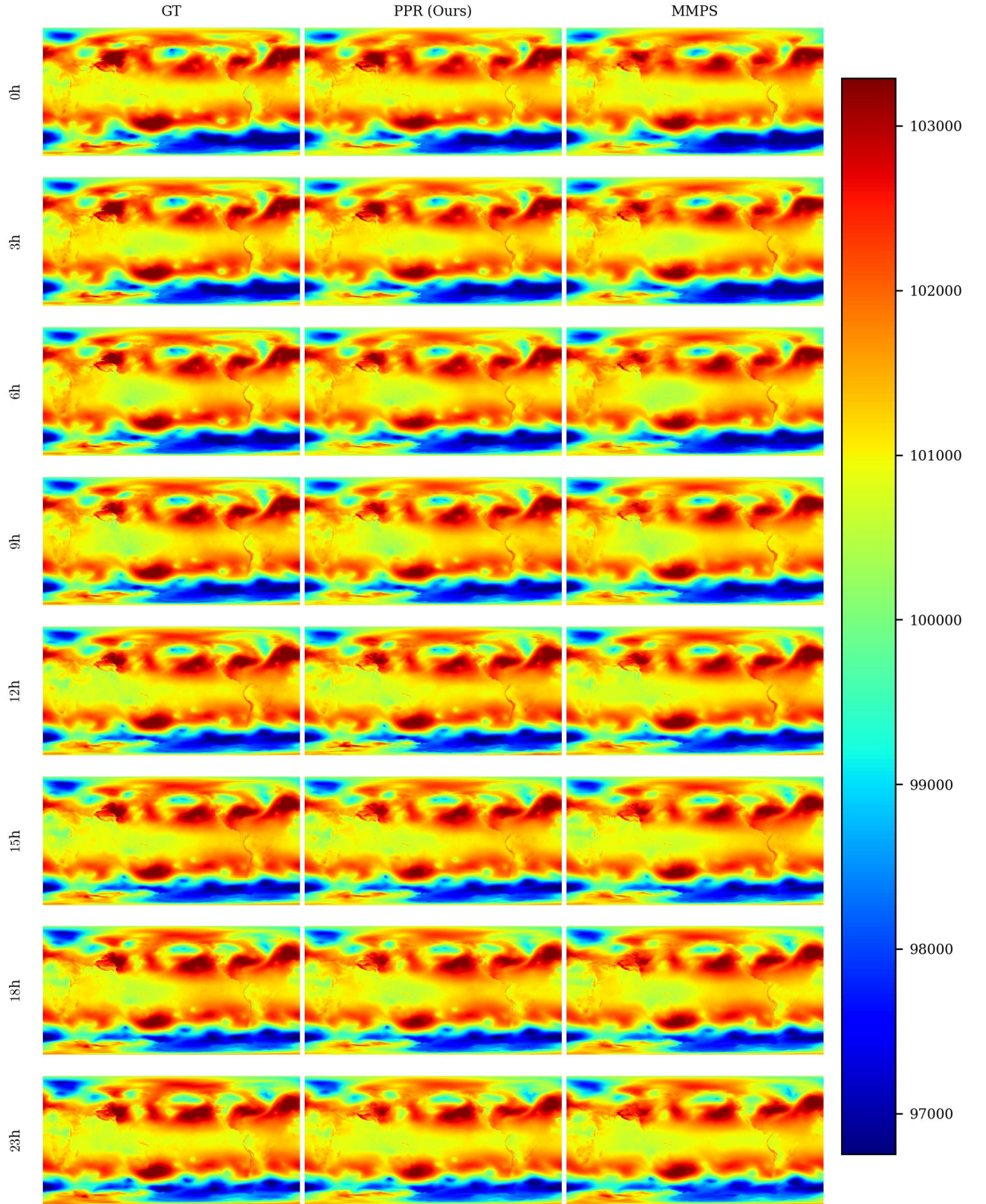


Figure 5.21. WEATHER sample showing the mean sea level pressure (Pa) for the ground truth, PPR, and MMPS. The y-axis is the hour in 24 h format.



## 5.G EXAMPLE SAMPLES FROM ALL EXPERIMENTS

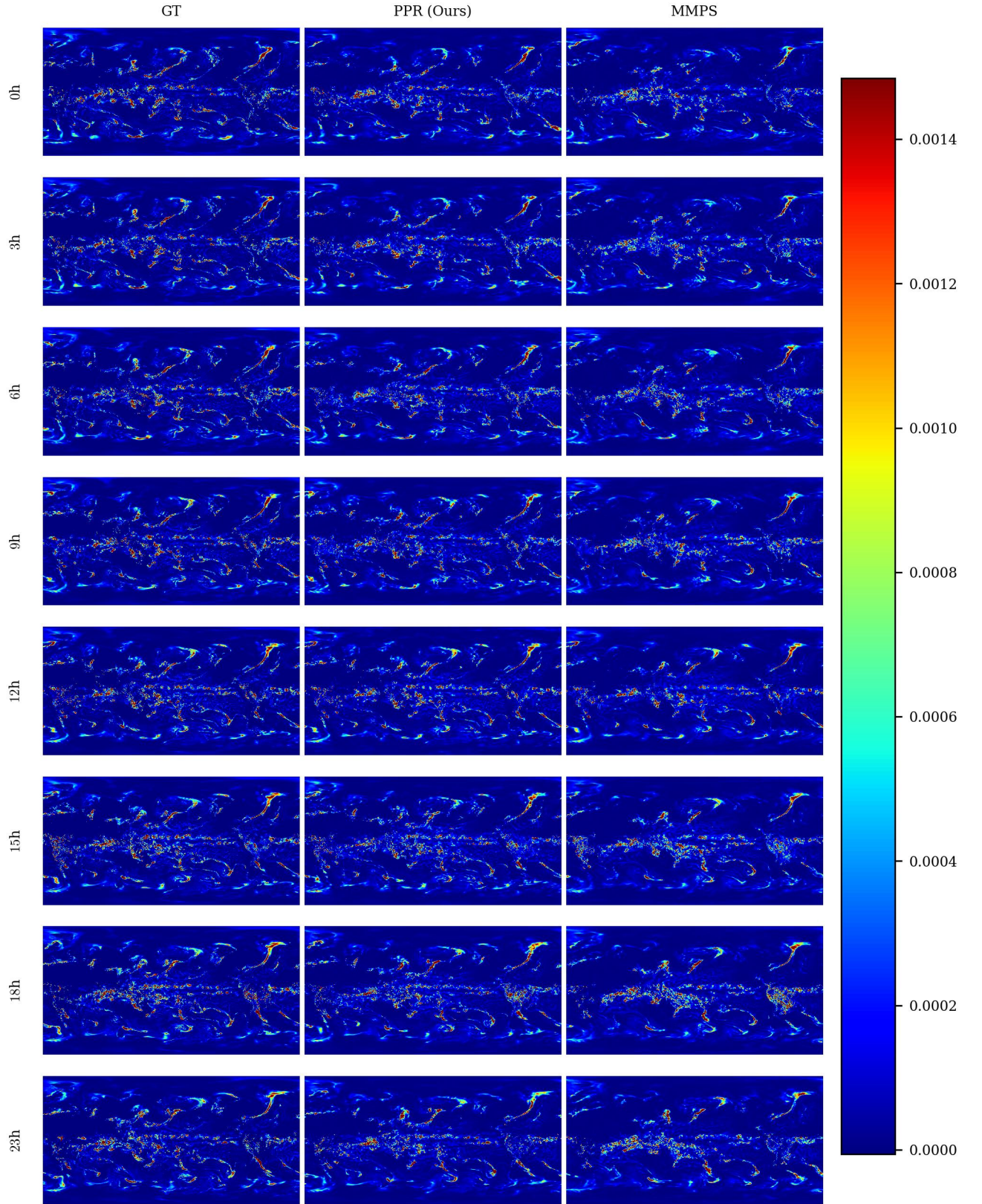


Figure 5.22. WEATHER sample showing the total precipitation (m) for the ground truth, PPR, and MMPS. The y-axis is the hour in 24 h format.

# III

## EPILOGUE

## 6 CONCLUSION

*Notoriamente no hay clasificación del universo que no sea arbitraria y conjetural.  
La razón es muy simple: no sabemos qué cosa es el universo.*

— Borges, *El idioma analítico de John Wilkins*

This thesis set out to make data-driven models of physical systems cheaper to train and run, more faithful to the physics they are meant to represent, and usable for inference at the scale of real systems. The four chapters that precede this one each address part of that goal. What follows summarizes what was done and then argues for a position, one that is close to the opposite of the one I held when I started. I also record a few personal thoughts about the field and about future work, without any claim to their correctness, and offer some closing remarks.

### 6.1 SUMMARY OF CONTRIBUTIONS

**Efficiency (RQ1).** Chapter 2 presents NeuralMPM, which reformulates the emulation of Lagrangian particle systems as a grid-based problem. Rather than building a graph over particles and learning message passing on it, we interpolate particles onto a fixed Eulerian grid, compute the dynamics with standard grid-to-grid networks, and interpolate back. Combined with temporal bundling and short autoregressive rollouts during training, this reduces training times from days to hours and memory consumption by one to two orders of magnitude relative to graph-network emulators, while matching or exceeding their long-horizon accuracy across water, sand, goop, and multi-material datasets. Chapter 3 addresses the same bottleneck at a very different scale, together with the problem of posterior inference. A global atmospheric state at  $0.25^\circ$  resolution has on the order of  $10^8$  components, which puts diffusion directly in state space out of reach. Appa learns an autoencoder that compresses trajectories by a factor of over 500 and trains a score-based model in that latent space, making generative modeling of the global atmosphere feasible on an academic compute budget.

**Consistency and sampling (RQ2).** Chapter 4 studies training-free, post hoc projections for enforcing governing-equation constraints on neural PDE solvers. Treating the network output as an initialization and solving a constrained optimization problem anchored at it, we find that nonlinear projections reduce residual violation by orders of magnitude while *improving* accuracy over physics-informed training. We also characterize when this works: the anchor must be a reasonable approximation, and linearized variants are locally effective but degrade away from the linearization point. Chapter 5 carries the idea into the generative setting, where the target is no longer a single prediction but a distribution. The central observation is that correct constrained sampling requires approximating the constrained marginals  $p_t^C$  induced by running the forward diffusion from the constrained set, not merely projecting the final sample. We

define the constrained forward process, derive its marginals and backward process, identify a corrector kernel whose unique stationary distributions are those marginals, and design Predict-Project-Renoise (PPR) to approximate it. Each of its three components is shown to be necessary, and PPR obtains higher feasibility rates, lower constraint violations, and a better distribution of mass than competing training-free methods.

**Scale (RQ3).** Appa is a latent weather diffusion model that can serve as a prior in the Bayesian sense. Because it models trajectories unconditionally, it can be conditioned at inference time on arbitrary observation operators, so that reanalysis, filtering, and forecasting are not separate models but special cases of posterior sampling from a single prior. Classical variational assimilation achieves large-scale conditioning on heterogeneous observations through tangent-linear and adjoint models, but ordinarily returns a single optimized analysis rather than posterior samples [1, 2]. Simulation-based inference does not currently reach this regime [3, 4]. Chapter 5 then shows that constrained sampling remains viable at that scale, using Appa itself as one of its test-beds.

## 6.2 ON PHYSICAL CORRECTNESS

I began this thesis thinking that physical correctness could be recovered after the fact. That position is attractive: it decouples constraint satisfaction from training, it applies to models one did not train and cannot retrain, and it makes the constraint an explicit object that can be inspected and swapped. Two chapters of building such methods have moved me to the opposite view. Correctness should be learned during training, from data, and post hoc enforcement should be understood as a repair for models that failed to learn it. The argument has two halves. The practical one is that post hoc enforcement is gated on knowledge one frequently does not have, at a recurring cost that grows with the realism of the system, and that when that knowledge is only approximate, the output is misaligned with the data. The philosophical one is that physical correctness is a property of the data itself, and belongs where the data is modeled rather than where it is sampled.

*... in the first paragraph something struck him that he had not observed: he muttered “Il faut que j’y songe encore”, and put the paper in his pocket.*

— De Morgan on Lagrange, *A Budget of Paradoxes*, 1872

The practical half turns on verification, and the contrast with code shows what physical domains lack. It is by now close to a cliché to note the recent progress in large language models and coding agents, but it is nevertheless true, and one reason for it is that code is cheap to verify. The weakest form of verification is to run the program and ask whether it crashes (halting problem aside); a stronger one is to run it and compare its output against an expected answer; a stronger one still is a test suite that somebody else wrote. Whatever human effort they require is amortized: once written, none of them need a human in the loop again, all of them are cheap relative to the cost of producing the program, and all of them can be applied at scale to samples of unknown quality. A signal with those properties can be turned into a benchmark [5, 6] and, more importantly, into a training objective [7, 8].

*Program testing can be used to show the presence of bugs, but never to show their absence!*

— Dijkstra, *Notes on Structured Programming*, 1970

Physical domains offer no comparable oracle. Nature is governed, at a very basic level, by symmetries [9], and there is a large body of work on building them into machine learning models [10]. But a symmetry is a necessary condition, not a sufficient one. Physical states respect the symmetries of the system that produced them, but respecting those symmetries does not make a state physical: a field can be divergence-free, equivariant under rotation, and correctly periodic, and still be something the system could never occupy. Symmetries constrain the candidate set, but do not characterize it.

The other natural place to look is differential equations, since most of what we know



about the physical world is written in their language<sup>7</sup>. What makes them expressive is also what makes them awkward as an oracle. The laws of nature are easiest to state locally, as relations between derivatives<sup>8</sup>, with trajectories, or states in the case of static problems, recovered by integration. That integration is the difficult part. Numerical solution of nonlinear PDEs is notoriously delicate, and the fine spatial and temporal discretizations it demands are the reason surrogates are attractive in the first place [11–13]; it is equally well known that different discretizations of the same equation can produce visibly different trajectories [14, 15]. A state therefore cannot be checked against a governing equation in the abstract. It has to be discretized in some particular way first, but doing so can approach the cost of solving the system in the first place. That raises the question of whether the compute is well spent, since the same budget could be given to more training data, to longer rollouts, or to a larger model. In toy systems the question is academic, the residual being a handful of finite differences on a small state. In realistic ones it is not, and a verification step that costs an appreciable fraction of a solve erodes the speedup that motivated learning a surrogate at all.

Cost, however, is the benign part of the problem. The deeper one is that the residual we write down is generally not the one the data satisfies. Conservation laws are the favorable case, since they are typically known, and yet they need not hold on a finite simulation domain. The Well dataset [16], for instance, contains stellar simulations in which mass and energy leave the system by radiation or ejection into space, and therefore leave the domain. The law itself is intact, since the total mass-energy of the universe is conserved; it is the truncation of the domain that breaks it. For governing-equation residuals the situation is worse, because a PDE residual is not a property of the equation alone, but of the equation *together with a discretization*. The data a surrogate is trained on was produced by a simulator with its own spatial and temporal discretization, its own flux limiters and subgrid closures, possibly implicit in time and adaptive in space, and its own forcing terms, which might not even have been saved. That operator is usually unavailable, not recoverable from the data, and sometimes not fixed. What one writes down instead is a convenient residual, typically cheap finite differences on a uniform grid. It is not the operator that generated the trajectories, and driving it to zero does not recover the data manifold; it moves samples onto a different one, the manifold that is self-consistent under the wrong operator. The projection is then a bias at best, and a real limitation at worst. Even when the physics is known, in other words, the residual may not be.

And the physics is not always known. Governing equations come with unresolved scales, parameterized closures, and phenomena left out of the model, so partial physics is the normal case rather than the exception [17]. Verification against partial physics is not merely incomplete, it is misguided: a well-generated state, or a real measurement, can be flagged as incorrect, and a method that trusts the flag will move it somewhere worse. An oracle that rejects valid states is not a weak oracle; it is a different oracle from the one we wanted. Methods whose preconditions are met only in the regime where the problem was already easy are not a general answer.

The other half of the argument is philosophical. Physical correctness is a property of the support of the data. It holds for every state the system can occupy, before any measurement is taken and independently of which measurement is taken. The Bayesian view makes this clearer still: in an inverse or a design problem, the posterior has to be physical, and it is more natural for that physicality to come from the prior than from an artificial separate likelihood added after the fact to account for physical correctness. Appa is a case in point: we have no analytical expression, and not even a full understanding, of what makes a given atmospheric state valid, yet we want its posteriors—reanalysis, filtering, or forecasting, depending only on what is conditioned upon—to be physical.

<sup>7</sup> *Physical reality has been thought of as represented by continuous fields, governed by partial differential equations.*

— Einstein, 1931

<sup>8</sup> *Given an equation involving any number of fluent quantities, to find the fluxions, and vice versa.*

— Newton, *Epistola posterior* to Leibniz, 1676

*Since all models are wrong the scientist must be alert to what is importantly wrong. It is inappropriate to be concerned about mice when there are tigers abroad.*

— Box, *Science and Statistics*, 1976

Methods like PPR should therefore not be the mechanism by which a model becomes physical. Their role is to repair the prior, and to impose the constraints that apply to a subset of the data distribution rather than to all of it, which are the constraints a prior could not have learned in the first place. For the time being that role does not disappear. Learning is never exact, a model trained to respect a structure will respect it only approximately, and some correction at sampling time will therefore remain useful. There may be a bitter lesson [18] here as well, with the need to repair the prior shrinking as data and compute grow, and hopefully vanishing eventually.

## 6.3 FUTURE DIRECTIONS

**Learning constraints and physicality from data.** The most important direction, and the one the previous section leads to, is to stop assuming that the constraint is given, or that physical correctness can be verified analytically. If physical structure is to be respected by the model rather than imposed on its outputs, and if the analytic residual is unknown, mis-specified by its discretization, or too expensive to evaluate, then the structure itself has to come from data. Concretely, this means learning the set on which trajectories lie rather than the equation that is supposed to define it, learning corrections or invariants directly from simulator output so that whatever discretization produced the data is absorbed rather than guessed at, and designing architectures for which the relevant structure holds by construction, so that no enforcement is needed at any stage. This last route is rewarding, since those models cannot represent infeasible data, but it requires knowledge of the structure. Perhaps the answer is simply more data and compute.

**Validating physicality of distributions.** Learning the structure sharpens the question the previous section started from, rather than dissolving it: how would we know that what was learned is right? Current practice reports violation of the residual we chose, which certifies consistency with our own assumptions rather than fidelity to the system, and no closed form is available to check a learned constraint against. The difficulty is greater still for generative models, where the object to be validated is a distribution and the errors can be statistical rather than pointwise. Progress here would look like tests built on quantities that survive discretization, on invariances checked against held-out data, and on calibration assessed in the same terms as accuracy. The uncertainty-quantification question raised by Appa is a special case: an inference system whose posteriors are not validated produces numbers of unknown reliability, however good they look. The general case of evaluating high-dimensional distributions is an infamous open problem.

**Beyond equality and differentiability.** Both projection chapters of this thesis stop at constraints that are equalities and that expose a gradient. Inequality and mixed constraints are the common case in practice, and they admit natural treatments, such as projections that terminate on entering the feasible set. Discrete and black-box constraints, which arise whenever the constraint is a legacy solver or a combinatorial condition, require a different mechanism entirely. Extending the framework in these directions would widen its reach considerably, and would also show whether the guarantees of Chapter 5 survive when the constraint offers no gradient.

## 6.4 CLOSING REMARKS

Data-driven modeling of physical systems has moved quickly from demonstrations to systems that are competitive with, and occasionally better and more flexible than, the numerical methods they were meant to imitate [19–21]. Some of the field’s momentum



has gone into scale: larger models, larger datasets, broader coverage. That effort has been vindicated by results, and while there is still a long way to go, it is not the part that worries me. What has not kept pace is our ability to say what these models guarantee and to check whether they deliver it. Physical correctness is still mostly asserted through partial or weak metrics [22], and the gap becomes more consequential as the models leave research settings.

This thesis contributes to the first kind of progress and, I hope, a little to the second. Its methods make emulation cheaper, make inference possible at scales it had not reached, and make constrained sampling correct in a sense that can be stated precisely. Its conclusion is more modest than the position I started from: enforcing physics after the fact works, and it works well when we know what we are enforcing, but knowing what we are enforcing is the hard part, and it is not a problem that enforcement can solve. The structure has to come from the data, because in the end that is the only place it undoubtedly exists.

*The best material model of  
a cat is another, or  
preferably the same, cat.*

— Rosenblueth and Wiener,  
*The Role of Models in  
Science*, 1945

## REFERENCES

- [1] P. Courtier, J.-N. Thépaut, and A. Hollingsworth. “A Strategy for Operational Implementation of 4D-Var, Using an Incremental Approach”. In *Quarterly Journal of the Royal Meteorological Society* 120.519 (1994), pp. 1367–1387.
- [2] R. N. Bannister. “A review of operational methods of variational and ensemble-variational data assimilation”. In *Quarterly Journal of the Royal Meteorological Society* 143.703 (2017), pp. 607–633.
- [3] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. “The frontier of simulation-based inference”. In *Proceedings of the National Academy of Sciences* 117.48 (2020).
- [4] Joeri Hermans et al. “A Trust Crisis In Simulation-Based Inference? Your Posterior Approximations Can Be Unfaithful”. In *Transactions on Machine Learning Research* (2022).
- [5] Mark Chen et al. “Evaluating Large Language Models Trained on Code”. 2021.
- [6] Carlos E. Jimenez et al. “SWE-bench: Can Language Models Resolve Real-World GitHub Issues?” 2023.
- [7] Nathan Lambert et al. “Tulu 3: Pushing Frontiers in Open Language Model Post-Training”. 2024.
- [8] DeepSeek-AI. “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning”. 2025.
- [9] Emmy Noether. “Invariante Variationsprobleme”. In *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* (1918), pp. 235–257.
- [10] Michael M. Bronstein et al. “Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges”. 2021.
- [11] Dmitrii Kochkov et al. “Machine Learning-Accelerated Computational Fluid Dynamics”. In *Proceedings of the National Academy of Sciences* 118.21 (2021), e2101784118.
- [12] Zongyi Li et al. “Fourier Neural Operator for Parametric Partial Differential Equations”. In *International Conference on Learning Representations*. 2021.
- [13] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. “Message Passing Neural PDE Solvers”. In *International Conference on Learning Representations*. 2022.
- [14] Yohai Bar-Sinai et al. “Learning data driven discretizations for partial differential equations”. 2019.
- [15] Randall J. LeVeque. “Finite Volume Methods for Hyperbolic Problems”. Cambridge Texts in Applied Mathematics. Cambridge: Cambridge University Press, 2002.
- [16] Ruben Ohana et al. “The Well: a Large-Scale Collection of Diverse Physics Simulations for Machine Learning”. In *Advances in Neural Information Processing Systems*. Vol. 37. 2024.
- [17] Benjamin Sanderse et al. “Scientific Machine Learning for Closure Models in Multiscale Problems: A Review”. In *Foundations of Data Science* 7.1 (2025), pp. 298–337.
- [18] Richard S. Sutton. “The Bitter Lesson”. 2019.
- [19] Remi Lam et al. “Learning Skillful Medium-Range Global Weather Forecasting”. In *Science* 382.6677 (2023), pp. 1416–1421.

- [20] Ilan Price et al. “Probabilistic Weather Forecasting with Machine Learning”. In *Nature* 637 (2025), pp. 84–90.
- [21] Cristian Bodnar et al. “A Foundation Model for the Earth System”. In *Nature* 641 (2025), pp. 1180–1187.
- [22] Nick McGreivy and Ammar Hakim. “Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations”. 2024.