Computers and Fluids 314 (2026) 107098

Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid

Physics-aware generative models for turbulent fluid flows through energy-consistent stochastic interpolants☆

Nikolaj T. Mücke [a],*, Benjamin Sanderse [a,b]

[a] Scientific Computing, Centrum Wiskunde & Informatica, Science Park 123, Amsterdam, 1098 XG, The Netherlands
[b] Centre for Analysis, Scientific Computing and Applications, Eindhoven University of Technology, PO Box 513, Eindhoven, 5600 MB, The Netherlands

ARTICLE INFO

Keywords:
Stochastic interpolants
Generative models
Stochastic differential equations
Fluid dynamics
Energy conservation
Turbulence

ABSTRACT

Generative models have demonstrated remarkable success in domains such as text, image, and video synthesis. In this work, we explore the application of generative models to fluid dynamics, specifically for turbulence simulation, where classical numerical solvers are computationally expensive. We propose a novel stochastic generative model based on stochastic interpolants, which enables probabilistic forecasting while incorporating physical constraints such as energy stability and divergence-freeness. Unlike conventional stochastic generative models, which are often agnostic to underlying physical laws, our approach embeds energy consistency in a soft manner by making the parameters of the stochastic interpolant learnable coefficients. We evaluate our method on a benchmark turbulence problem – Kolmogorov flow – demonstrating superior accuracy and stability over state-of-the-art alternatives such as autoregressive conditional diffusion models (ACDMs) and PDE-Refiner. Furthermore, we achieve stable results for significantly longer roll-outs than standard stochastic interpolants. Our results highlight the potential of physics-aware generative models in accelerating and enhancing turbulence simulations while preserving fundamental conservation properties.

1. Introduction

Recently developed generative models for scientific applications [1–5] mark a ground-breaking era in simulating complex systems. A natural question is therefore to what degree generative models can be used to support or replace simulation codes that have been classically used to model problems arising in domains like physics, chemistry, or biology. In particular, our interest lies in generative models for fluid dynamics problems. Fluid dynamics simulation codes have been developed in the last decades based on known physical principles (e.g. conservation laws), but with the main limitation that they are computationally very expensive to run. Pre-trained generative models offer promising alternatives for physics simulations by significantly accelerating computations. These models also provide uncertainty quantification and can model unresolved scale processes [6].

Neural network surrogate models, trained to replace PDEs within certain configurations, are a first step in this direction [7–9]. Later, more widely applicable "foundation models" have been developed for e.g. weather prediction [10–13]. These foundation models [14] are very large neural networks (sometimes having more than 1 billion parameters [10]), that are pre-trained on vast (re-analysis) datasets, and can be finetuned to specific tasks such as regional or seasonal forecasting. Such models have shown the potential to accelerate classical numerical weather predictions to the point that a 5-day forecast can be made in less than a minute [10]. The foundation model approach has also been applied to computational chemistry [15], biology [16], and fluid mechanics [17–23]. In [23], a foundation model was pre-trained on certain PDEs (compressible Euler and incompressible Navier–Stokes) and gave accurate results on PDEs that were not in the pre-training set.

These foundation models are pre-trained, but unlike their counterparts in text, video and image generation, they are still usually deterministic in nature and mainly focus on forecasting the mean of the possible trajectories [3,10,13,23]. This can lead to blurred forecast states [13,24]. A few stochastic machine learning approaches have recently been introduced to address this issue [3,25,26]. Moreover, stochasticity naturally arises when developing reduced models of fluid dynamics problems, in which the state is typically split into large (resolved) and small (unresolved) scales, and the purpose is to infer models for only the large scales. An example is through the Mori–Zwanzig formalism, in which the effect of the initial condition of the unresolved scales is typically modeled by a noise term. Similarly, one can argue that the evolution of the large scales is *not* deterministic, even when the evolution of the full model is deterministic.

☆ This article is part of a Special issue entitled: 'fusing data and physics' published in Computers and Fluids.

* Corresponding author.
E-mail address: nikolaj.mucke@cwi.nl (N.T. Mücke).

https://doi.org/10.1016/j.compfluid.2026.107098
Received 1 April 2025; Received in revised form 30 January 2026; Accepted 17 April 2026
Available online 24 April 2026

This manifests itself in non-uniqueness of the problem (different small scale realizations can have the same effect on the large scales), which warrants the use of a stochastic approach [27]. We note here that, outside the realm of machine learning, stochastic approaches have been long in use to model unresolved processes, typically known as 'stochastic parameterizations' [6,28]. The stochastic approach that we are considering is in line with those approaches, but benefits from the advanced approximation capabilities of neural networks.

Previous research has explored the approximation of SDEs using neural networks. In [29] neural networks are trained with gradients computed via adjoint methods. While such methods alleviate some of the computational burden compared with naively backpropagating through a solver, it is still prohibitively expensive for very high-dimensional problems. Specifically, systems of up to 50 dimensions are studied in [29]. In [1] similar approaches are utilized for learning a neural SDE as a closure term for LES simulations. To minimize the computational restrictions of high-dimensional systems, the discrete state dimension is reduced via an autoencoder and the SDE is trained in the latent space. While this approach is indeed promising, the use of autoencoders makes it difficult to infuse physics knowledge into the model due to the non-physical nature of the latent space. Furthermore, this approach relies on an LES solver, as the neural network only approximates a closure term. It is unclear how this approach would fare as a full non-intrusive surrogate model. In this work, as an alternative to the expensive adjoint-based training methods, we will draw inspiration from the field of generative models.

The current state-of-the-art in stochastic generative models for fluid flows relies mainly on denoising diffusion probabilistic models (DDPMs) [2–4] – similar to what is used in machine learning models for image generation [30,31], video generation [32], and diffusion language models [33]. DDPMs are based on sampling from a Gaussian distribution and transforming it through a sequence of steps into an image resembling one from the data distribution. In the context of time-dependent problems, one deals with a sequence of time steps, and a realistic 'image' (e.g. a flow field) needs to be created *at each time step*. This significantly complicates the process compared to image generation. An example of how this can be achieved is given in [3]: sample from a Gaussian at each time step, denoise, and condition on the state at the previous time step. A similar approach is used in [2]. Denoising at each time step is rather involved and not in line with the actual physical process. In general, a common issue with existing stochastic generative models is that they are agnostic of the underlying physical processes that are being modeled. For example, conservation of mass and energy are fundamental physical principles in fluid flows, but are not obeyed by existing generative models. This can lead to issues with the stability of predictions, especially when considering long prediction horizons.

In this work, we develop a pre-trained, stochastic generative model for stable simulation of physical processes. Our approach is built upon two key components: stochastic interpolants and energy stability. Stochastic interpolants, introduced in [34,35], possess the crucial ability to bridge two arbitrary probability distributions over a finite time interval. This property enables their use in time-dependent simulations for time stepping, as demonstrated in [5], without requiring the sampling and transformation of Gaussian distributions [2,3].

The main novelty of our work lies in the second component: energy stability through the imposition of a carefully designed stochastic interpolant (and, consequently, the drift term). Energy stability plays a dual role: it not only improves numerical stability [36,37], but also promotes physical correctness in the generative process. By designing an interpolant that respects energy conservation, we ensure that the drift term is trained on physically consistent data, leading to generated samples that softly adhere to this constraint. See Fig. 1 for a visual representation of our energy-consistent stochastic interpolant framework. Our work can be seen as a new approach to achieve stability with neural SDEs, which is known to be difficult to achieve with neural ODEs, requiring either specialized equation forms [36], specialized filters [38], clipping [39], noise addition [40], or online learning [41] (for an overview, see [42]).

This paper is structured as follows. In Section 2, we introduce the problem setting and provide an overview of the stochastic interpolant framework for probabilistic forecasting. Section 3 details our proposed modifications to the stochastic interpolant framework, emphasizing energy consistency, divergence-freeness, and efficient sampling. Implementation details, including the neural network architecture, training procedures, and inference strategies, are discussed in Section 4. In Section 5, we evaluate our methodology on a Kolmogorov flow test case and compare its performance with state-of-the-art alternatives. Finally, in Section 6, we summarize our findings and outline potential directions for future research.

2. Preliminaries

Stochastic variables are denoted with capital letters, X, and time-dependent stochastic variables, X_t, are denoted with a subscript. We assume the existence of a probability density function, p, associated with the probability measure, P, and will refer to the density when referring to the underlying distribution. Furthermore, to simplify notation we identify the densities through the arguments rather than by using subscripts. Thus, we will write $X \sim p(x)$ when X is distributed according to $P(X)$ and $X \sim p(x|y)$ when X is distributed according to $P(X|Y)$.

2.1. Problem setting

In this work we are interested in stochastic approaches to approximate the solution of (deterministic) PDEs, and in particular the incompressible Navier–Stokes equations:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla p + \frac{1}{\mathrm{Re}} \nabla^2 \boldsymbol{u} + \boldsymbol{f}(\boldsymbol{u}), \tag{1}$$

supplemented with the divergence-free constraint

$$\nabla \cdot \boldsymbol{u} = 0. \tag{2}$$

Here $\boldsymbol{u}(\boldsymbol{x}, t)$ is the velocity field, $p(\boldsymbol{x}, t)$ is the (scaled) pressure, and Re is the Reynolds number. Discretization in space and time leads to the fully discrete equation

$$\boldsymbol{u}_h^{n+1} = \boldsymbol{F}_h \left(\boldsymbol{u}_h^n; \mu \right). \tag{3}$$

Here $\boldsymbol{u}_h^n \in \mathbb{R}^D \approx \boldsymbol{u}(\boldsymbol{x}, t^n)$ and $\boldsymbol{F}_h$ is the discrete right-hand side incorporating both the spatial and temporal discretization, and μ represents physical parameters such as the Reynolds number and parameterized initial and boundary conditions. We assume that the discretized model can resolve all relevant scales of motion present in (1), at the cost of computation time. Therefore, we refer to Eq. (3) as the high-fidelity model, also known as Direct Numerical Simulation (DNS).

The high-fidelity model is often computationally prohibitive to solve, and a common approach is to reduce the range of scales present in (3) by a model reduction step. For the case of the Navier–Stokes equations, a common model reduction step that we will also employ in this work is *filtering*. Filtering aims to remove the smallest scales of the flow through a low-pass filter A, $\boldsymbol{u}_h^* := A\boldsymbol{u}_h$, such that $\boldsymbol{u}_h^* \in \mathbb{R}^d$ can be represented and simulated on a much coarser grid than $\boldsymbol{u}_h$ ($d \ll D$) [43]. However, $\boldsymbol{u}_h^*$ does not satisfy the Navier–Stokes equations because the filter does not commute with the PDE operations.

A main ongoing challenge is thus to find a parameterized evolution equation that approximates the (exact) large scales $\boldsymbol{u}_h^*$ [42], i.e. an equation of the form

$$\bar{\boldsymbol{u}}_h^{n+1} = \boldsymbol{G}_{h,\theta} \left(\bar{\boldsymbol{u}}_h^n; \mu \right), \tag{4}$$

such that $\bar{\boldsymbol{u}}_h \approx \boldsymbol{u}_h^*$ and θ are learnable parameters. Many existing approaches formulate the parametric model $\boldsymbol{G}_{h,\theta}$ in terms of $\boldsymbol{F}_h$ with

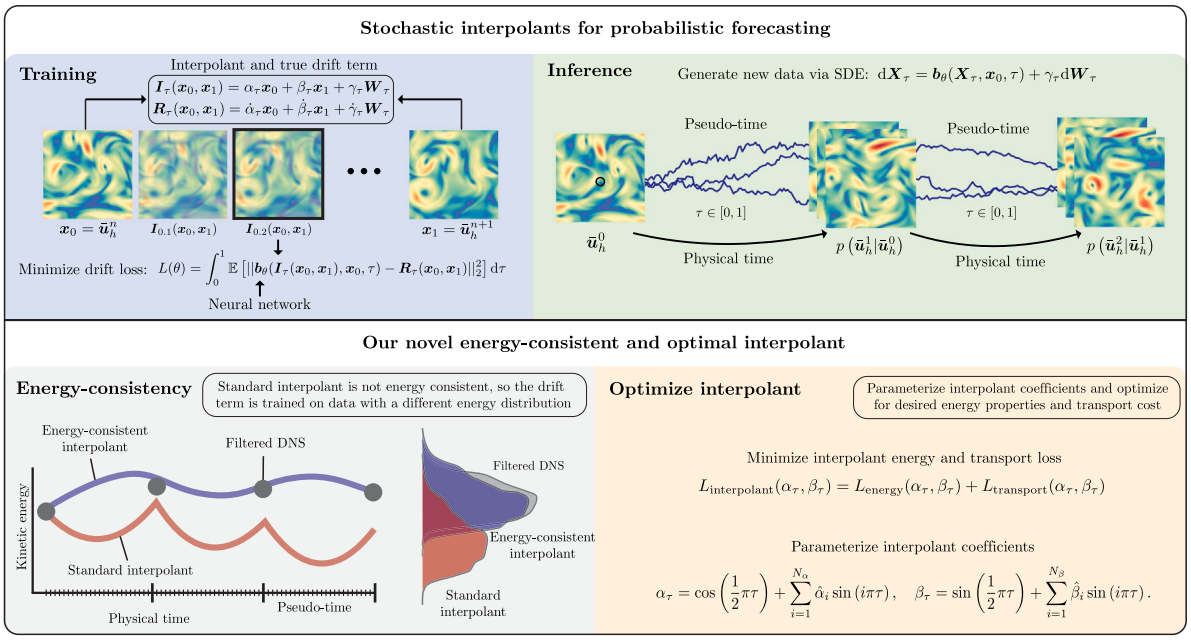


Fig. 1. Visualization of the various steps and components in the energy-consistent stochastic interpolant framework. At the top, the existing framework presented in [5] is visualized. The training is performed by sampling two consecutive physical states and interpolating between them via the stochastic interpolant in pseudo-time. The interpolant is used to train a drift term in an SDE that will be solved in pseudo-time during inference to generate new states conditioned on the initial state. Choosing the interpolant without including physics knowledge can result in inconsistencies in energy with respect to the physical energy, as visualized in the lower left part of the figure. In this paper, we propose to optimize the interpolant for energy-consistency by minimizing a loss over the interpolant coefficients as shown in the lower right part of the figure.

an additional correction term, also known as a closure model [42]. Usually, these models are deterministic, which corresponds to the fact that Eq. (1) is deterministic. However, there are good reasons to model the reduced dynamics instead with a stochastic approach as highlighted in the introduction [27,44,45]. SDEs are therefore a natural fit for turbulence, and neural SDEs are in particular interesting, given their combination of stochasticity, time continuity, and the expressive capability of neural networks. So far, neural SDEs seem under-explored for turbulent flows, most likely due to the costs of training and simulating stochastic systems. We address this issue through recent developments in training SDE-based generative models, and in particular by using so-called stochastic interpolants (SIs). For some very recent related contributions that use DDPMs, see [46–48].

In our stochastic approach we replace the deterministic state, $\bar{\boldsymbol{u}}_h^n$, with a stochastic variable, $\bar{\boldsymbol{U}}_h^n$. Due to the stochastic nature, the time evolution of the state can be rewritten in terms of sampling from a conditional distribution:

$$\bar{\boldsymbol{U}}_h^{n+1} \sim p(\bar{\boldsymbol{u}}_h^{n+1}|\bar{\boldsymbol{u}}_h^n). \tag{5}$$

We typically assume that the initial density, $p(\bar{\boldsymbol{u}}_h^0)$, is known or that the initial condition is simply given $\bar{\boldsymbol{U}}_h^0 = \bar{\boldsymbol{u}}_h^0$. The conditional distribution in (5) is generally not available and is approximated via ensembles. The task is thus to find a stochastic version of (4) that produces ensemble members that are distributed according to $p(\bar{\boldsymbol{u}}_h^{n+1}|\bar{\boldsymbol{u}}_h^n)$.

2.2. Stochastic interpolants for probabilistic forecasting

In this section, we present the principles of the generative model for probabilistic forecasting trained via the stochastic interpolant (SI) framework. The aim is to generate samples from the conditional distribution, $p(\bar{\boldsymbol{u}}_h^{n+1}|\bar{\boldsymbol{u}}_h^n)$, via a stochastic differential equation (SDE) that transforms samples from a base distribution to samples from the target distribution. In the standard version of the SI framework, the SDE is not trained to mimic the physical evolution of the state. It should rather be interpreted as a means of transforming samples from one distribution to another. Hence, the SDE is not solved with respect to physical time, t, discussed in the previous section, but it is solved in pseudo-time, τ, introduced with the sole purpose of facilitating the transformation. The pseudo-time interval can therefore be chosen freely, but is typically chosen to be $[0, 1]$ for simplicity.

SIs are a type of generative models introduced in [34] and expanded upon in [5,35]. The general idea is similar to SDE-based denoising diffusion models where a normal distribution is being transformed into the target distribution. In the SI framework, however, one can transform samples from an arbitrary distribution into samples from another arbitrary distribution. This property makes it a suitable choice for physics-based modeling. In this section, we summarize the findings from [5], which focuses on utilizing SIs for probabilistic forecasting.

Consider the densities $p(\boldsymbol{x}_0)$ and $p(\boldsymbol{x}_1|\boldsymbol{x}_0)$, with $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathbb{R}^d$. Given a sample $\boldsymbol{x}_0$, the aim of the generative model is to sample from the conditional distribution:

$$p(\boldsymbol{x}_1|\boldsymbol{x}_0) = \frac{p(\boldsymbol{x}_0, \boldsymbol{x}_1)}{p(\boldsymbol{x}_0)} > 0. \tag{6}$$

With the SI framework, such samples are generated by solving an SDE with initial condition $\boldsymbol{x}_0$ on the pseudo-time interval $\tau \in [0, 1]$:

$$\mathrm{d}\boldsymbol{X}_\tau = \boldsymbol{b}_\theta(\boldsymbol{X}_\tau, \boldsymbol{x}_0, \tau)\mathrm{d}\tau + \gamma_\tau \mathrm{d}\boldsymbol{W}_\tau, \quad \tau \in [0, 1], \quad \boldsymbol{X}_0 = \boldsymbol{x}_0. \tag{7}$$

The drift term, $\boldsymbol{b}_\theta$, is modeled as a neural network with weights, θ. The SI framework provides a way of training it such that solutions of the SDE at $\tau = 1$, are distributed according to the conditional distribution, $\boldsymbol{X}_{\tau=1} \sim p(\boldsymbol{x}_1|\boldsymbol{x}_0)$. The diffusion term γ_τ[1] and Wiener process $\boldsymbol{W}_\tau$ form the source of stochasticity. In this work $\gamma_\tau = 0.1(1-\tau)$ is chosen as recommended in [5]. As mentioned earlier, the SDE is solved in pseudo-time, τ, in contrast to the physical time, t.

[1] Not to be confused with the diffusion term $\nu\nabla^2\boldsymbol{u}$ in the Navier–Stokes equations.

The key element in training the drift term is the so-called stochastic interpolant:

$$\boldsymbol{I}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) = \alpha_\tau \boldsymbol{x}_0 + \beta_\tau \boldsymbol{x}_1 + \gamma_\tau \boldsymbol{W}_\tau = \boldsymbol{x}_\tau. \tag{8}$$

The Wiener process, $\boldsymbol{W}_\tau$, can be sampled at a specific pseudo-time point, τ_i, with $\boldsymbol{W}_{\tau=\tau_i} = \sqrt{\tau_i}\boldsymbol{z}$, where $\boldsymbol{z} \sim N(0,1)$. There is freedom in the choice of the τ-dependent functions α_τ, β_τ, and γ_τ, but they need to satisfy the temporal boundary conditions, $\alpha_0 = \beta_1 = 1$ and $\alpha_1 = \beta_0 = \gamma_1 = 0$. This ensures that $\boldsymbol{I}_{\tau=0}(\boldsymbol{x}_0, \boldsymbol{x}_1) \sim p(\boldsymbol{x}_0)$ and $\boldsymbol{I}_{\tau=1}(\boldsymbol{x}_0, \boldsymbol{x}_1) \sim p(\boldsymbol{x}_1|\boldsymbol{x}_0)$. The dynamics of the interpolant in Eq. (8) given an initial condition $\boldsymbol{x}_0$ can be written as

$$\mathrm{d}\boldsymbol{I}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) = \underbrace{\left(\dot{\alpha}_\tau \boldsymbol{x}_0 + \dot{\beta}_\tau \boldsymbol{x}_1 + \dot{\gamma}_\tau \boldsymbol{W}_\tau\right)}_{:=\boldsymbol{R}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1)} \mathrm{d}\tau + \gamma_\tau \mathrm{d}\boldsymbol{W}_\tau, \quad \tau \in [0,1], \quad \boldsymbol{X}_0 = \boldsymbol{x}_0, \tag{9}$$

where $\dot{(\cdot)}$ denotes differentiation with respect to τ. Hence, solving the SDE from 0 to τ can be seen as a mapping of a samples, $\boldsymbol{x}_0$, to a sample from $\boldsymbol{X}_\tau \sim p(\boldsymbol{x}_\tau|\boldsymbol{x}_0)$. In particular, solving Eq. (9) from 0 to 1 maps a sample, $\boldsymbol{x}_0$, to a sample from $\boldsymbol{X}_1 \sim p(\boldsymbol{x}_1|\boldsymbol{x}_0)$. Solving the SDE in Eq. (9) requires that both $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are available. However, the goal is to generate samples, $\boldsymbol{x}_1$, when only having access to $\boldsymbol{x}_0$. Therefore, we train $\boldsymbol{b}_\theta$, which only takes in $\boldsymbol{x}_0$, τ, and the intermediate state, $\boldsymbol{X}_\tau$, to match $\boldsymbol{R}_\tau$. With proper training the solution of Eq. (7) approximates the solution of Eq. (9), $\boldsymbol{X}_\tau \approx \boldsymbol{I}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1)$ for all τ, including the endpoint of interest, $\boldsymbol{X}_1 \approx \boldsymbol{I}_1(\boldsymbol{x}_0, \boldsymbol{x}_1) = \boldsymbol{x}_1$. This enables generation of samples from $p(\boldsymbol{x}_1|\boldsymbol{x}_0)$ by using $\boldsymbol{x}_0$ alone via Eq. (7).

Given choices of α_τ, β_τ, γ_τ, and training data $(\boldsymbol{x}_0, \boldsymbol{x}_1) \sim p(\boldsymbol{x}_0, \boldsymbol{x}_1)$, the interpolant can be evaluated and the drift term is trained by minimizing the following loss function with respect to the model weights, θ [5]:

$$L(\theta) = \int_0^1 \mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{z})} \left[\left\| \boldsymbol{b}_\theta(\boldsymbol{I}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1), \boldsymbol{x}_0, \tau) - \boldsymbol{R}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) \right\|_2^2 \right] \mathrm{d}\tau. \tag{10}$$

$\|\cdot\|_2$ denotes the standard l^2-norm. During training, the integral in (10) is approximated by sampling τ uniformly. The expected value is approximated using the samples $(\boldsymbol{x}_0, \boldsymbol{x}_1) \sim p(\boldsymbol{x}_0, \boldsymbol{x}_1)$ from a training set and samples from the Wiener process are sampled via $\boldsymbol{W}_\tau = \sqrt{\tau}\boldsymbol{z}$, $\boldsymbol{z} \sim N(0,1)$. In practice this is minimized by sampling mini-batches of $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ and τ. We found that sampling a single τ-value for each pair $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ did not hurt the training when compared to sampling several τ-values for each state pair. It is important to note that this setup enables training a model to approximate a drift term via a simple mean squared error-type loss without ever solving any SDEs during the training stage. See Alg. 1 for pseudo-code of the training stage. Note that Alg. 1 is a simplified algorithm. In practice, there are additional considerations to be taken into account, and we refer to Section 4.2 for details.

An important property of the SI framework is that we learn the drift term of a *continuous* SDE that generates new samples. This means that one can choose an SDE solver and number of pseudo-time steps after training. Such choices determine the computational time and the quality of the samples. This trade-off between accuracy and computation time can be made based on the application at hand.

After training, new samples, $\boldsymbol{X}_1 \sim p(\boldsymbol{x}_0|\boldsymbol{x}_1)$, can be generated by solving (7) with the trained drift term and an appropriate SDE solver. In particular, by choosing $\boldsymbol{x}_0 = \bar{\boldsymbol{u}}_h^n$ and $\boldsymbol{x}_1 = \bar{\boldsymbol{u}}_h^{n+1}$ the model learns to sample from the distribution of interest, $p(\bar{\boldsymbol{u}}_h^{n+1}|\bar{\boldsymbol{u}}_h^n)$ – namely the distribution of the next physical state given the current state. This operation represents a single application of the generative model. The transition from $\bar{\boldsymbol{u}}_h^n$ to $\bar{\boldsymbol{u}}_h^{n+1}$ is referred to as a *physical time step*, denoted by superscripts. This notation and terminology serve to distinguish physical time from pseudo-time (τ), which is indicated by subscripts. Then, by setting $\boldsymbol{x}_0 = \bar{\boldsymbol{u}}_h^{n+1}$ and solving (7) again, we can obtain samples from $p(\bar{\boldsymbol{u}}_h^{n+2}|\bar{\boldsymbol{u}}_h^{n+1})$. Thus, we can obtain arbitrarily long trajectories in the *physical* space by repeatedly solving the trained SDE, provided the solution is stable. Typically, at each physical time step we solve the SDE in pseudo-time several times in order to get an ensemble representation of the distribution. A visualization of this process is given at the top of Fig. 1.

Algorithm 1: Training drift term in stochastic interpolant framework

Input: α_τ, β_τ, γ_τ, untrained $\boldsymbol{b}_\theta$, training data, N_{epochs}, Optimizer
for $i = 1 : N_{epochs}$ **do**
  for $(\boldsymbol{x}_0, \boldsymbol{x}_1)$ *in training data* **do**
    Sample pseudo time, $\tau \sim U[0,1]$;
    Sample Wiener process, $\boldsymbol{W}_\tau = \sqrt{\tau}\boldsymbol{z}$, $\boldsymbol{z} \sim N(0,1)$;
    Evaluate interpolant, $\boldsymbol{I}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) = \alpha_\tau \boldsymbol{x}_0 + \beta_\tau \boldsymbol{x}_1 + \gamma_\tau \boldsymbol{W}_\tau$;
    Evaluate $\boldsymbol{R}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) = \dot{\alpha}_\tau \boldsymbol{x}_0 + \dot{\beta}_\tau \boldsymbol{x}_1 + \dot{\gamma}_\tau \boldsymbol{W}_\tau$;
    Compute drift loss, $L(\theta)$, via Eq. (10);
    Update drift model weights: $\theta \leftarrow$ Optimizer$(\theta, L(\theta))$;
  end for
end for
Output: Trained $\boldsymbol{b}_\theta$

The SI framework for probabilistic time stepping does not necessarily generate physically plausible trajectories. As an example, we visualize the kinetic energy of the state as a function of physical time, as well as the distribution of the energy for a Kolmogorov flow approximated with the SI framework in Fig. 2. In the Kolmogorov flow the energy should remain constant in expectation. However, we see that after 300–400 physical time steps, the energy starts to grow. This is reminiscent of instabilities that have been reported when using neural networks and neural ODEs to represent turbulence [36,38,49].

Furthermore, the distributions clearly do not match. Solving the SDE with more time steps improves this slightly, but not enough to a satisfactory degree. In the following section, we outline improvements to the SI setup that mitigates these issues. For more details on the Kolmogorov flow, see Section 5.

3. A new energy-consistent stochastic interpolant

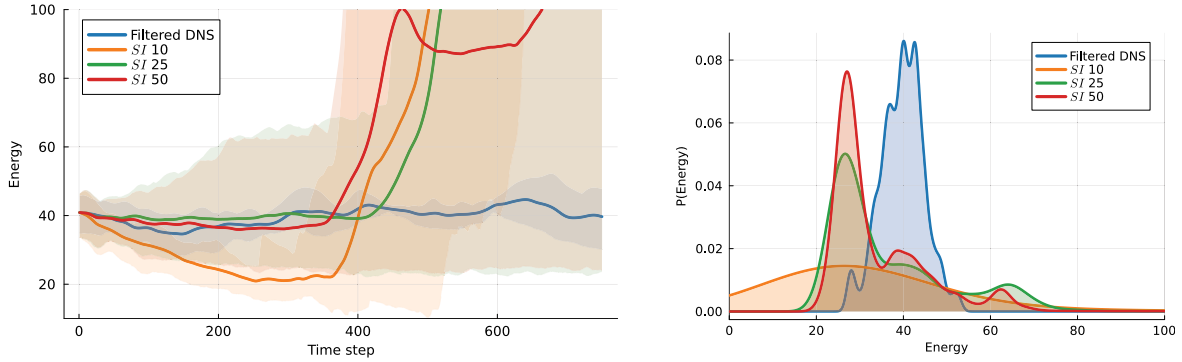
3.1. Physics consistency in generative models

With proper training and neural network architecture, it is theoretically possible to generate samples that resemble the true conditional distribution, including quantities of interest such as energy. However, in practice this is not an easy task due to various sources of errors including, but not limited to, suboptimal training resulting in local minima, insufficient training data limiting generalization to unseen data, and numerical integration errors leading to inaccurate SDE solutions. These potential errors become even more detrimental during long roll-outs where errors can accumulate, leading to inaccurate or unstable simulations, similar to what has been observed for neural ODEs [50]. This is largely due to the time-dependent nature of the problem, which gives an additional complexity that is not present in image generation. While this is theoretically alleviated with bigger neural networks and more training, it is in practice difficult to achieve the long-term stability with such approaches alone.

As a consequence, with current SIs only short roll-outs have been considered. For example, [5] considered roll-outs of two physical steps. In this paper, we aim to generate significantly longer trajectories by promoting stability through physical consistency of the SI.

To achieve physical consistency in the generation procedure there are several steps that can be modified in the SI framework. Firstly, the interpolant in Eq. (8) can be adjusted to have desired properties. Secondly, the loss function in Eq. (10) can be extended. Thirdly, the neural network architecture used to approximate the drift term can be designed with physics consistency in mind. Lastly, the generative SDE in Eq. (7) can be modified, to impose desired properties after training.

In general, physical consistency can either be imposed as *soft* or *hard* constraints. When imposing soft constraints, one typically adds



(a) Time evolution of kinetic energy (b) Kinetic energy distributions.

Fig. 2. Energy results when simulating Kolmogorov flow with the stochastic interpolants as presented in Section 2.2. (a) shows the energy evolution of the physical state and (b) shows the energy distribution of an ensemble of states over multiple time steps and trajectories. SI 10, SI 25, and SI 50 refer to the generated trajectories using 10, 25, and 50 SDE steps in the SI SDE respectively.

additional terms to the loss function during training. Although this does not guarantee exact adherence to the constraints, it is often easier to implement. Imposing hard constraints, on the other hand, ensures adherence to the constraint. This is typically done via the architecture [51] or additional computations in shape of projections [38], constraint optimization [52], or by modifying the equations of interest [36]. The choice between hard constraints or soft constraints depends on the problem at hand. Additional computations sometimes associated with hard constraints can lead to high computational costs. A specific architecture might make the training easier and the predictions more stable, but it could also impose limitations as one does not allow the neural network to potentially find an optimal representation. In addition, in a stochastic setting, there are additional considerations to be taken into account. The system under consideration might only conserve certain properties in *distribution*, while other properties are conserved for all *realizations*.

For the specific case of the incompressible Navier–Stokes Eqs. (1)–(2), arguably the most important physical structures are the divergence-freeness of the flow field and the conservation of kinetic energy (in the absence of boundaries and viscosity) [36,37,53]. This is further detailed in Appendix A. Our aim is therefore to ensure that the generated trajectories are *energy-consistent* and *divergence-free*. Energy-consistency will be promoted in distribution via a soft constraint which determines the parameterization of the SI in Section 3.2. Divergence-freeness will be enforced via a hard constraint while time-stepping the SDE in Section 3.3.

3.2. Energy-consistent interpolant

The interpolant defines the stochastic paths between $\bar{\boldsymbol{u}}_h^n$ and $\bar{\boldsymbol{u}}_h^{n+1}$. The drift term is trained directly on the interpolated states, such that the generating SDE approximates these paths. Therefore, the specific choice of interpolant plays a major role in the training and by extension also the generation. When generating data without any immediate need for physics-consistency, it typically does not matter if these paths adhere to any physical laws. However, when physics-consistency is of importance, the choice of interpolant should adhere to the desired properties so the model is trained on physics-consistent data.

As mentioned above, some properties should only be enforced on average. Energy-consistency is one such property where individual trajectories should not necessarily adhere to the constraint, but an ensemble of realizations should be energy-consistent (see Appendix A). Therefore, we choose to impose energy-consistency in a *soft* manner. In this section, we show how one can optimize the choice of α_τ and β_τ to achieve such properties.

The following theorem forms the basis of the optimization:

Theorem 3.1 (*Energy Evolution of the Interpolant*). *For a stochastic interpolant defined as in Eq. (8) and energy defined by* $\frac{1}{2}\left\|\boldsymbol{I}_\tau\right\|_2^2$, *the time evolution of the interpolant energy is given by:*

$$\mathrm{d}\left[\frac{1}{2}\left\|\boldsymbol{I}_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1)\right\|_2^2\right] = (\boldsymbol{I}_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1)\cdot\boldsymbol{R}_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1)+\frac{d}{2}\gamma_\tau^2)\mathrm{d}\tau+\gamma_\tau\boldsymbol{I}_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1)\cdot\mathrm{d}\boldsymbol{W}_\tau. \tag{11}$$

Furthermore, the expected time evolution is given by:

$$\mathbb{E}_{(\boldsymbol{x}_0,\boldsymbol{x}_1,\boldsymbol{W})}\left[\mathrm{d}\left(\frac{1}{2}\left\|\boldsymbol{I}_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1)\right\|_2^2\right)\right] = \mathbb{E}_{(\boldsymbol{x}_0,\boldsymbol{x}_1)}\left[H_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1;\alpha_\tau,\beta_\tau,\gamma_\tau)\right]\mathrm{d}\tau, \tag{12}$$

where

$$\begin{aligned}H_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1;\alpha_\tau,\beta_\tau,\gamma_\tau) &= \dot{\alpha}_\tau\alpha_\tau\left\|\boldsymbol{x}_0\right\|_2^2+\dot{\beta}_\tau\beta_\tau\left\|\boldsymbol{x}_1\right\|_2^2\\ &+(\dot{\beta}_\tau\alpha_\tau+\dot{\alpha}_\tau\beta_\tau)\langle\boldsymbol{x}_0,\boldsymbol{x}_1\rangle+\dot{\gamma}_\tau\gamma_\tau\tau d+\frac{d}{2}\gamma_\tau^2.\end{aligned} \tag{13}$$

See Appendix B for the proof of the theorem. Note that the energy depends on the grid size, h. As h is assumed constant, it does not change any of the results above nor the derivations below besides a scalar multiplication. See Appendix C for more details.

With Eq. (12), we can control the expected rate of change of the kinetic energy of the SI through the choice of its parameters α_τ and β_τ. Despite γ_τ being present in Eq. (12), and thereby affecting the energy evolution, we do not choose to control the expected rate of change via γ_τ. α_τ and β_τ determine the mean flow from $\boldsymbol{x}_0$ to $\boldsymbol{x}_1$ and γ_τ only controls the noise levels. Therefore, α_τ and β_τ have a greater effect on the trajectory. In earlier stages of this research, we conducted experiments including γ_τ in the optimization and we did not see improved results, confirming our intuition. Furthermore, as presented in [5], the diffusion term in the generating SDE does not have to be γ_τ, but can be tuned after training. Including γ_τ in the fitting of the interpolant coefficients would limit that flexibility. Future research will examine how adjusting noise levels post-training influences the energy of generated samples.

The key idea is that in many physical systems we have a-priori knowledge about the rate of change of kinetic energy between $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ – see Appendix A for the Navier–Stokes equations. This knowledge can be used to determine α_τ and β_τ. Denoting the known rate of change by $k_\tau(\boldsymbol{x}_0,\boldsymbol{x}_1)$, the loss function quantifying the expected discrepancy

between the desired rate of change and the actual rate of change induced by the interpolant reads:

$$L_{\text{energy}}(\alpha_\tau, \beta_\tau) = \int_0^1 \mathbb{E}_{(\boldsymbol{x}_0,\boldsymbol{x}_1)} \left[\left| H_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1; \alpha_\tau, \beta_\tau, \gamma_\tau) - k_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) \right|^2 \right] \mathrm{d}\tau$$
$$= \int_0^1 \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \left| H_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1; \alpha_\tau, \beta_\tau, \gamma_\tau) - k_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) \right|^2 p(\boldsymbol{x}_0, \boldsymbol{x}_1) \mathrm{d}\boldsymbol{x}_0 \mathrm{d}\boldsymbol{x}_1 \mathrm{d}\tau$$
$$\approx \frac{1}{N_\tau N_s} \sum_{i=1}^{N_\tau} \sum_{j=1}^{N_s} \left| H_\tau(\boldsymbol{x}_{0,j}, \boldsymbol{x}_{1,j}; \alpha_{\tau_i}, \beta_{\tau_i}, \gamma_{\tau_i}) - k_\tau(\boldsymbol{x}_{0,j}, \boldsymbol{x}_{1,j}) \right|^2 . \tag{14}$$

In this expression, τ_i, $i = 1, \dots, N_\tau$ are pseudo-time samples between 0 and 1, and $\boldsymbol{x}_{0,j}$ and $\boldsymbol{x}_{1,j}$, $j = 1, \dots, N_s$ are samples from the training set. In our simulations we are interested in the case where energy is conserved on average. This corresponds to choosing $k_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) = 0$, optimizing the interpolant such that the change in energy is zero in expectation.

Minimizing $L_{\text{energy}}(\alpha_\tau, \beta_\tau)$ with respect to α_τ and β_τ requires parameterizing α_τ and β_τ. In order to do so, we write α_τ and β_τ as Fourier series and optimize with respect to the coefficients, $\hat{\alpha}_i$ and $\hat{\beta}_i$:

$$\alpha_\tau = \cos\left(\frac{1}{2}\pi\tau\right) + \sum_{i=1}^{N_\alpha} \hat{\alpha}_i \sin(i\pi\tau), \quad \beta_\tau = \sin\left(\frac{1}{2}\pi\tau\right) + \sum_{i=1}^{N_\beta} \hat{\beta}_i \sin(i\pi\tau). \tag{15}$$

Note that these expressions satisfy the temporal boundary conditions by construction. While Fourier series have been chosen for this study, they are not the only option. Other basis functions, such as Legendre or Chebyshev polynomials are also a potential option, as well as nonlinear representations such as neural networks, similar to what is considered in [54]. Further investigation of these approaches will be pursued in future work. Choosing α_τ and β_τ to minimize (14) is effectively a *soft* constraint on the training data: on average, the resulting trajectories will conserve energy, but each individual trajectory can still have locally increasing or decreasing energy. This is in line with the properties of the incompressible Navier–Stokes equations with body force (see Appendix A).

The minimization of L_{energy} with respect to $\hat{\alpha}_\tau$ and $\hat{\beta}_\tau$ will be performed in a pretraining step, i.e., before the training of the drift term as described in Section 2.2. The optimization problem to be solved is of dimension $N_\alpha + N_\beta$. Our experiments indicate that $N_\alpha = N_\beta < 10$ is sufficient. Hence, higher-order optimization methods, such as Newton methods, can be used without computational issues.

The general idea of optimizing the interpolant coefficients is similar to what is presented in [54]. The three key differences are that (i) in [54] they optimize a parameterized noise schedule in a diffusion model jointly with the neural network weights, (ii) they parameterize the noise schedule as a neural network, and (iii) they only have the standard diffusion model loss and no additional targets like physics consistency in this paper. The difference in target is natural as they consider image generation and not physical processes. We have chosen Fourier series instead of neural networks due the simplicity, number of trainable parameters, and smoothness properties. However, we expect that similar results can be achieved by parameterizing α_τ and β_τ as neural networks in the context of this paper. The choice of separating the training of the interpolant coefficients and the neural network in this work is due to the differences between the standard SI objective and the physics consistency objective. Firstly, the interpolant coefficient optimization is a simpler problem with significantly fewer parameters to be fitted, which allows for higher order optimization methods such as Newton methods. Secondly, by decoupling the two stages we avoid the issues of weighing the two objectives which could result in poor performance if not done properly.

3.2.1. Minimizing path complexity

If one only minimizes the energy discrepancy, there is a chance that the resulting interpolant can be complex and include high-frequency oscillations. Therefore, we also want to promote low *complexity* in the trajectories. In this section we describe further improvements to be made to the interpolant like an additional loss term for fitting the α_τ and β_τ. The aim is to reduce the complexity of the paths defined by the interpolant connecting $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$. In this context, "complexity" refers to factors that make the paths difficult to learn and integrate. Hence, reducing complexity must simplify the training stage, and reduce the number of necessary pseudo-time steps when generating new data. To this end, we use the transport cost as a metric for complexity. In [34] the connection between the SI framework for normalizing flows and the optimal transport problem in the framework of [55] is established. The transport cost is defined by [55]:

$$C_{\text{transport}}(\boldsymbol{R}_\tau) = \int_0^1 \mathbb{E}_{(\boldsymbol{x}_0,\boldsymbol{x}_1)} \left[\left\| \boldsymbol{R}_\tau(\boldsymbol{x}_1, \boldsymbol{x}_0) \right\|_2^2 \right] \mathrm{d}\tau$$
$$= \int_0^1 \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \left\| \boldsymbol{R}_\tau(\boldsymbol{x}_1, \boldsymbol{x}_0) \right\|_2^2 p(\boldsymbol{x}_1, \boldsymbol{x}_0) \mathrm{d}\boldsymbol{x}_0 \mathrm{d}\boldsymbol{x}_1 \mathrm{d}\tau. \tag{16}$$

The transport cost measures the cost of transforming one distribution to another. Minimizing the transport cost minimizes the traveled distance between $p(\boldsymbol{x}_0)$ and $p(\boldsymbol{x}_1|\boldsymbol{x}_0)$.

In [34], it is briefly discussed how one can minimize the transport cost while training the drift term by solving a max–min problem. However, max–min problems are typically difficult to handle due to the saddle point structure of the optimum. In this paper, we take a slightly different approach and perform this optimization *before* training the drift term. By decoupling the training of the drift term and the identification of the energy-consistent interpolant we avoid solving a max–min problem which is generally more difficult to deal with. However, we now have to solve two optimization problems where the outcome of the first problem is used in the second. This does add some complexity in the implementation.

To minimize the transport cost, we use the same approach as discussed in Section 3.2. Using the parameterization from Eq. (15) we simultaneously minimize the transport cost as well as L_{energy}. This gives the full loss term for the interpolant:

$$L_{\text{interpolant}}(\alpha_\tau, \beta_\tau) = L_{\text{energy}}(\alpha_\tau, \beta_\tau) + L_{\text{transport}}(\alpha_\tau, \beta_\tau), \tag{17}$$

where

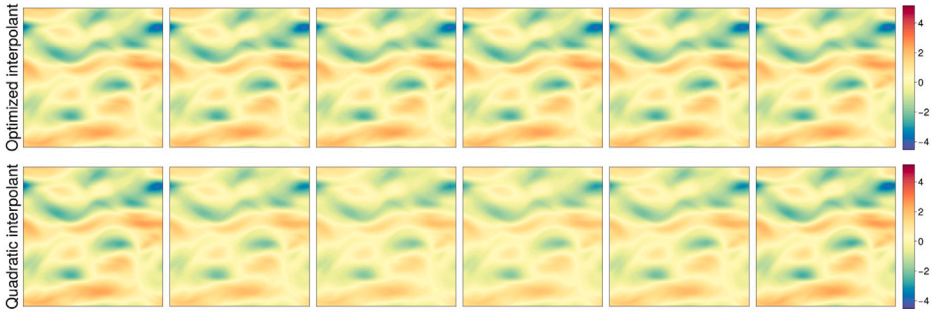
$$L_{\text{transport}}(\alpha_\tau, \beta_\tau) = \int_0^1 \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \left\| \boldsymbol{R}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) \right\|_2^2 p(\boldsymbol{x}_0, \boldsymbol{x}_1) \mathrm{d}\boldsymbol{x}_0 \mathrm{d}\boldsymbol{x}_1 \mathrm{d}\tau$$
$$= \int_0^1 \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \left\| \dot{\alpha}_\tau \boldsymbol{x}_0 + \dot{\beta}_\tau \boldsymbol{x}_1 + \dot{\gamma}_\tau \boldsymbol{W}_\tau \right\|_2^2 p(\boldsymbol{x}_0, \boldsymbol{x}_1) \mathrm{d}\boldsymbol{x}_0 \mathrm{d}\boldsymbol{x}_1 \mathrm{d}\tau$$
$$\approx \frac{1}{N_\tau N_s} \sum_{i=1}^{N_\tau} \sum_{j=1}^{N_s} \left\| \dot{\alpha}_{\tau_i} \boldsymbol{x}_{0,j} + \dot{\beta}_{\tau_i} \boldsymbol{x}_{1,j} + \dot{\gamma}_{\tau_i} \boldsymbol{W}_{\tau_i} \right\|_2^2, \tag{18}$$

with $(\boldsymbol{x}_{0,j}, \boldsymbol{x}_{1,j}) \sim p(\boldsymbol{x}_0, \boldsymbol{x}_1)$ for $j = 1, \dots, N_s$ being training samples. Note that $L_{\text{transport}}$ is simply $C_{\text{transport}}$ considered as a function of α_τ and β_τ instead of $\boldsymbol{R}_\tau$. See Algorithm 2 for the pseudo-code of the interpolant optimization procedure.

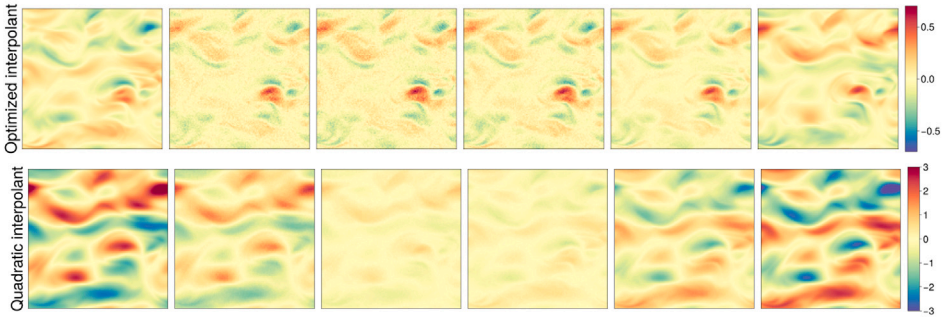
Note that the two terms, L_{energy} and $L_{\text{transport}}$, in Eq. (17) are not weighted. In this work we did not observe a necessity for this, as the two losses are dimensionally consistent and of the same order of magnitude. However, this might be different in other settings. In that case, a hyperparameter that weighs the two terms could be introduced. Similarly, a regularization term can be added if necessary.

In Fig. 3, we visually compare the quadratic interpolant proposed in [5], $\alpha_\tau = 1 - \tau$, $\beta_\tau = \tau^2$, with the interpolant optimized with respect to the transport and energy loss. In Fig. 3(a) it is apparent that the optimized interpolant and the quadratic interpolant are visually similar. However, the drift terms, $\boldsymbol{R}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1)$, vary quite a lot as shown in Fig. 3(b).

Fig. 4 compares various interpolant coefficients alongside their corresponding energy discrepancies and transport loss. Specifically, we



(a) Transport and energy optimized (top row) and quadratic (bottom row) interpolant, $\boldsymbol{I}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1)$, of the x-velocity for pseudo-time step from left to right: $\tau = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$.



(b) Transport and energy optimized (top row) and quadratic (bottom row) interpolant drift, $\boldsymbol{R}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1)$, of the x-velocity for pseudo-time step from left to right: $\tau = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$. Note the different limits in the colorbar.

Fig. 3. Comparison of the optimized interpolant and the interpolant proposed in [5] with $\alpha_\tau = 1 - \tau$ and $\beta_\tau = \tau^2$.

Algorithm 2: Optimize interpolant

Input: training data, N_α, N_β, N_{epochs}, Optimizer
for $i = 1 : N_{epochs}$ **do**
  Evaluate α_τ and β_τ via Eq. (15);
  Compute interpolant loss, $L_{\text{interpolant}}(\alpha, \beta)$, via Eq. (17);
  Update α_τ and β_τ: $(\alpha_\tau, \beta_\tau) \leftarrow \text{Optimizer}(\alpha_\tau, \beta_\tau, L(\alpha_\tau, \beta_\tau))$;
end for
Output: Trained α, β

compare our optimized results against the quadratic interpolant and the linear interpolant, used in the rectified flow formulation [56], with $\alpha_\tau = 1 - \tau$ and $\beta_\tau = \tau$. Note that the rectified flow is originally formulated for ODEs rather than SDEs. Moreover, the original paper does not address the specific case of sampling from a target distribution that is conditioned on the base. The comparison in Fig. 4 uses samples from Kolmogorov flow trajectories, where energy is constant on average. Hence, we set $k_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1) = 0$. While the optimized coefficients α_τ and β_τ appear visually similar to the linear formulation, they yield quite different performance metrics. Notably, the linear interpolant maintains strong energy conservation but incurs high transport costs, whereas the quadratic interpolant performs poorly across both metrics.

The optimized interpolants consistently achieve superior performance, whether the objective function only targets transport,[2] only targets energy, or both. As anticipated, joint optimization produces the most favorable results, yielding the best balance between lowest transport and energy conservation. These findings underscore the clear advantages of tailoring the interpolant via optimization rather than relying on fixed, heuristic formulations.

It is worth noting that despite the linear interpolant's strong energy conservation, the resulting SI model has been shown to perform poorly in practical applications (see [5] and Appendix D).

3.3. Divergence-consistency

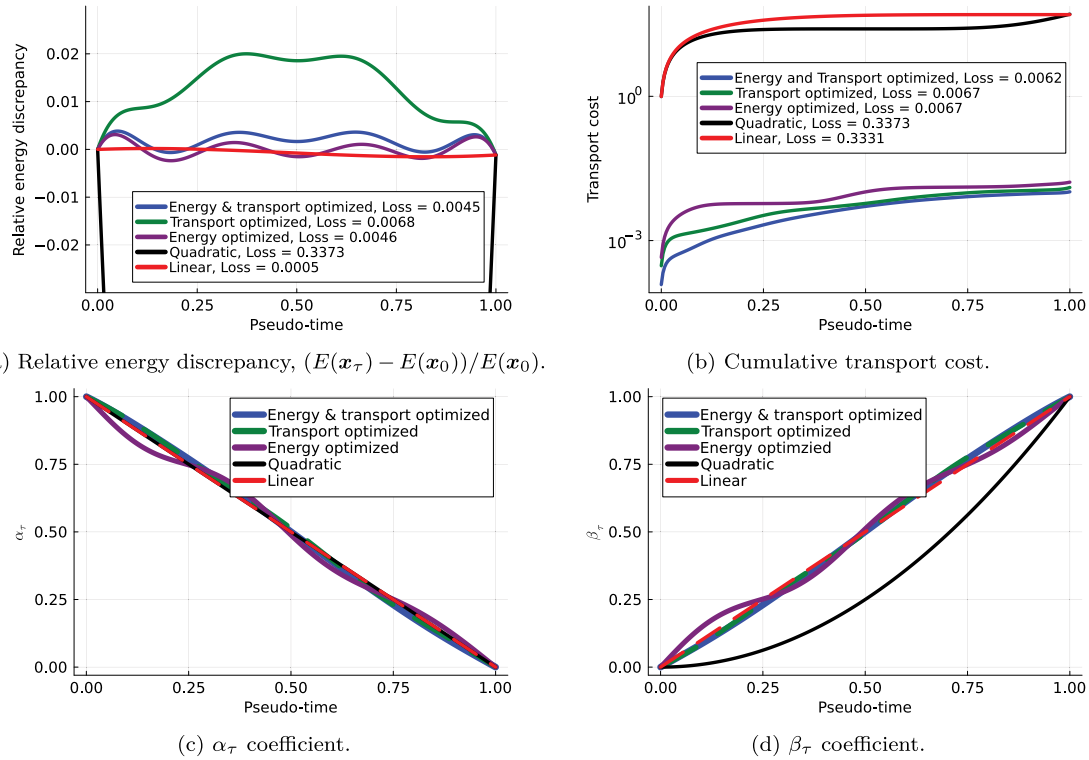
For incompressible fluid flows the velocity field is divergence-free. However, a generative model does generally not adhere to this property, despite being trained on divergence-free data. This is especially the case when dealing with long roll-outs due to accumulation of errors.

In contrast to energy-consistency, divergence-freeness in the incompressible Navier–Stokes equations is an algebraic constraint and not an evolution equation. In other words, every realization arising from solving the SDE in Eq. (7) should be divergence-free. For this reason we impose divergence-consistency as a hard constraint on every single realization.

Ensuring that the generated velocity fields are divergence-free can be done in several ways. For example, it can be done on the neural network architecture level [36]. Another approach is to learn an SDE for the stream function $\boldsymbol{\psi}$ instead of the velocity, such that $\boldsymbol{u} = \nabla \times \boldsymbol{\psi}$, and its divergence is by construction zero. As an alternative, we will make use of *projections* and the face-averaging filter as presented in [38], which are designed to keep the filtered velocity field divergence-free. The projection operator $\Pi : \mathbb{R}^d \to \mathbb{R}^d$, projects any field onto its divergence-free part. In [38] the projection is performed at every stage of the Runge–Kutta method by solving a Poisson equation. We use the same projection in this work — for details on Π, see Eqs. (A.8)–(A.9) in Appendix A.2.

Ideally, all interpolated states should be divergence-free both during training and inference. However, this would require projecting the state after every single pseudo-time step when solving the generating SDE.

[2] The transport-optimized interpolant was trained with l^2-regularization on the Fourier coefficients to ensure stability. This was found to be unnecessary for the other cases.



(a) Relative energy discrepancy, $(E(\boldsymbol{x}_\tau) - E(\boldsymbol{x}_0))/E(\boldsymbol{x}_0)$.

(b) Cumulative transport cost.

(c) α_τ coefficient.

(d) β_τ coefficient.

Fig. 4. Profiles of α_τ and β_τ optimized for energy conservation, transport cost, and both are shown. These are compared against the linear (rectified flow) and quadratic formulation proposed in [5]. Legends in (a) and (b) report the energy discrepancy and transport loss, respectively. Note that for the quadratic interpolant in (a), the energy discrepancy exceeds the plotted range, indicating significantly poorer conservation compared to the optimized and linear interpolants.

Algorithm 3: Trajectory generation with drift term learned via the stochastic interpolant

Input: Initial condition $\bar{\boldsymbol{u}}_h^0$, N_t, N_τ, $\boldsymbol{b}_\theta$, γ_τ, Π, SDE integrator
for $n_t = 0 : N_t$ **do**
  $\boldsymbol{X}_0 = \bar{\boldsymbol{u}}_h^{n_t}$;
  for $n_\tau = 0 : N_\tau$ **do**
    $\boldsymbol{X}_{n_\tau+1} = \text{SDE step}(\boldsymbol{X}_{n_\tau}, \boldsymbol{b}_\theta, \gamma_\tau)$;
  end for
  $\bar{\boldsymbol{u}}_h^{n_t+1} = \Pi \boldsymbol{X}_1$
end for
Output: $\{\bar{\boldsymbol{u}}_h^n\}_{n=1}^{N_t}$

As the projection requires solving a Poisson equation, it adds significant computational time. Therefore, we propose to project the state once during each physical time step, i.e. after the SDE has been solved in pseudo-time. Hence, the generation of samples is given by:

$$\mathrm{d}\boldsymbol{X}_\tau = \boldsymbol{b}(\boldsymbol{X}_\tau, \bar{\boldsymbol{u}}_h^n, \tau)\mathrm{d}\tau + \gamma_\tau \mathrm{d}\boldsymbol{W}_\tau, \quad \boldsymbol{X}_0 = \bar{\boldsymbol{u}}_h^n, \quad \tau = 0 \dots 1, \tag{19a}$$

$$\bar{\boldsymbol{u}}_h^{n+1} = \Pi \boldsymbol{X}_1. \tag{19b}$$

The projection operator is such that any (discretized) velocity field becomes divergence-free. Any noise present before projection will be affected by this step, and could be redistributed over the domain, due to the non-local nature of the projection operator. Hence, the projection is commenced regardless of the noise level.

4. Implementation

Besides the general framework presented in the previous sections, there are still several decisions regarding implementation to make. Here, we briefly discuss such considerations.

All implementations are done in Julia. The source code can be found on GitHub.[3] All trainings are performed on a single Nvidia RTX 3090 GPU.

4.1. Neural network architecture

The SI framework does not require a specific family of models to parameterize $\boldsymbol{b}_\theta$. However, due to the universality of neural networks, they are generally chosen for approximating the drift term. This choice of family of models naturally leads one to ask what architecture to use. In this regard, previous work on SIs for image generation have taken inspiration from work on DDPMs [5]. We do the same and take inspiration from work using DPPMs for fluid dynamics [46,57].

We make use of a UNet architecture, originally introduced in [58], with ConvNeXt layers instead of normal residual layers [59]. A sketch of the architecture is shown in Fig. 5. The pseudo-time, τ, is first embedded and then passed to the ConvNeXt layers as a bias term. It is embedded via a sinusoidal positional embedding followed by a shallow neural network and then passed onto the convolutional layers.

The downsampling is performed via strided convolutions and transposed convolutions are used for the upsampling. It is generally known that up- and downsampling via convolutional layers rather than pooling and interpolating gives better and less smoothed results [60,61].

In the bottleneck we use a diffusion transformer [62] to compute attention globally while also incorporating embedded time. We add a trainable positional encoding to the tensor before passing it to the diffusion transformer. The transformer layer ensures a global receptive field and the specific choice of diffusion transformer is made due to the efficient incorporation of parametric dependence.

[3] https://github.com/nmucke/StochasticInterpolants.jl.git

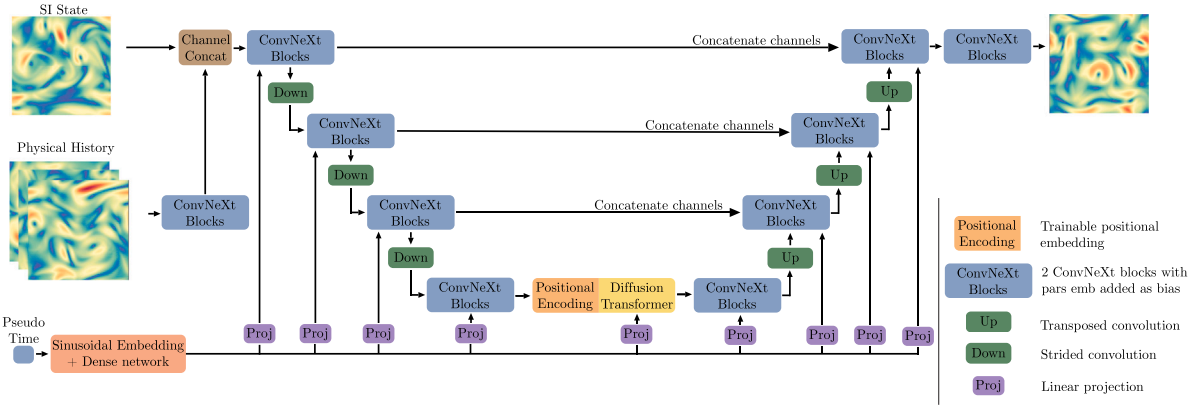


Fig. 5. Drift term neural network architecture.

Instead of only inputting the current physical state as conditions for the model, we make use of several previous states. While this has not been included in discussions and derivations in previous sections, it does not change the approach, as this only requires changing the conditioning in the drift term. It has been shown that providing a history as conditioning to the model and not just the current state results in higher accuracy [8,9,63]. The drift term and corresponding SDE in Eq. (7) change to:

$$\mathrm{d}\boldsymbol{X}_\tau = \boldsymbol{b}_\theta(\boldsymbol{X}_\tau, \bar{\boldsymbol{u}}_h^{n-l:n}, \tau)\mathrm{d}\tau + \gamma_\tau \mathrm{d}\boldsymbol{W}_\tau, \quad \boldsymbol{X}_0 = \bar{\boldsymbol{u}}_h^n, \quad \tau = 0 \dots 1, \tag{20}$$

where $\bar{\boldsymbol{u}}_h^{n-l:n} = (\bar{\boldsymbol{u}}_h^{n-l}, \dots, \bar{\boldsymbol{u}}_h^n)$ and l is the length of the history to be included.

Throughout the network we use the GELU activation function [64]. In the ConvNeXt layers and diffusion transformer, we use layer normalization. Lastly, we do not make use of dropout as experiments did not show improved performance.

See Fig. 5 for a visualization of the UNet architecture.

4.2. Training considerations

In this section we briefly describe the specific implementation of the training of the SI drift term and the interpolant coefficients.

The training states are standardized per channel. Each channel corresponds to a physical field, e.g. velocity in x-direction and y-direction. Each field is standardized to have zero mean and unit standard deviation. For the optimization, we use the AdamW optimizer [65] with a cosine annealing learning rate scheduler with a warmup [66]. The weights are regularized with L2-regularization. To further prevent overfitting, we make use of early stopping based on a validation dataset. The validation loss is computed by time-stepping via the generating SDE for multiple physical time steps. This ensures that the trained model actually performs well for the task it is intended for.

Optimizing the interpolant is performed using a batched Newton optimization algorithm with a backtracking line search for determining how much to update in the optimal search direction.

4.3. Inference

One of the key advantages of the SI method, and SDE-based generative models in general, is the flexibility it provides in the inference stage. As the drift term of a (continuous) SDE is learned, any numerical SDE solver can be used after training. Furthermore, the amount of pseudo-time steps used can also be chosen according to quality and time restrictions. In this work, we use the Heun SDE integrator [67].

5. Results

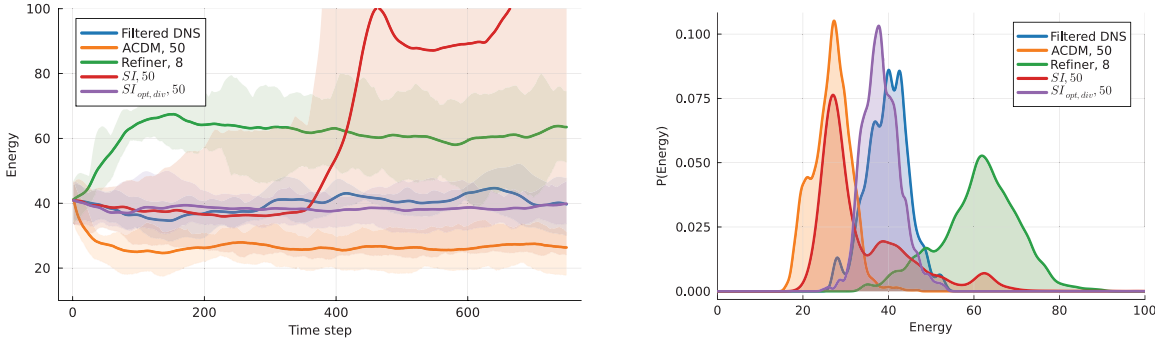
We present results for a Kolmogorov flow test case. The governing equations are the incompressible Navier–Stokes equations in two dimensions given by Eq. (1). For an overview of the test case settings, see Table 1.

We evaluate the proposed framework on a series of metrics and quantities of interest serving different purposes. We compute the mean squared error (MSE), the Pearson correlation, and LSiM [68], which measures how well the generated trajectories match the filtered DNS trajectories directly. Since the underlying dynamics are chaotic in nature, it is only to be expected that the generated trajectories match the filtered DNS trajectories in the short-term. Therefore, we also compute the kinetic energy, the energy spectrum, and the rate of change of the states, to assess if the generated trajectories have the same *characteristics* in terms of energy and rate of change as the filtered DNS trajectories. Such metrics are more suitable for assessing long-term behavior. Furthermore, we compare the distributions of some of these quantities to ensure that the statistics match. See Appendix C for details on the metrics and quantities of interest.

We compare our proposed methodology with three other approaches. Namely, the PDE-refiner [57] (referred to as Refiner from hereon), the Autoregressive Conditional Diffusion Model (ACDM) [46], and the original version of the SI for probabilistic forecasting without the proposed improvements [5]. These methods are generative models for probabilistic forecasting of states governed by PDEs. The Refiner and the ACDM utilize the DDPM framework. For the implementation of the ACDM and the Refiner, we use the code from the GitHub repository associated with [46].[4] The specific models used are taken directly from that repository and are slightly modified to approximately match the amount of model weights that we use for the stochastic interpolant. Note that a key difference between the ACDM and the Refiner compared to the SI framework is that the ACDM and Refiner need to be trained for a specific amount of diffusion steps. Hence, the number of generation steps must be chosen before training, unlike for the SI where the number of SDE generation steps can be chosen freely after training.

We generate 5 trajectories per test trajectory. That is, for a given initial condition, we generate 5 realizations with the generative models. All 5 realization are compared with the filtered DNS solution using the same initial condition. Since we are testing against 5 test trajectories, we generate a total of 25 realizations.

[4] https://github.com/tum-pbs/autoreg-pde-diffusion



(a) Time evolution of kinetic energy

(b) Kinetic energy distributions.

Fig. 6. Kinetic energy results for various generative models.

5.1. Kolmogorov flow

In this test case we assess the models ability to perform accurate and stable long-horizon simulations with respect to the statistics of the fluid flow.

Kolmogorov flow is a type of forced turbulent flow that obeys the Navier–Stokes equations. Specifically, we use the forcing:

$$\boldsymbol{f}(\boldsymbol{u}) = \sin(4y)\begin{bmatrix}1\\0\end{bmatrix} - 0.1\boldsymbol{u}. \tag{21}$$

We consider the domain, $\Omega = [0, 2\pi]^2$, and the time horizon $T = 62.5$. The first term in the forcing injects energy into the system and the second is a dissipative term that depends on the velocity. In total, the two terms ensure that the flow converges towards a statistically stationary state.

We simulate the Kolmogorov flow using a finite volume method on a staggered grid implemented in the IncompressibleNavierStokes.jl library [69].[5] The high-fidelity simulations are performed on a 2048 × 2048 grid and are downsampled to a 128 × 128 grid using face-averaging. We refer to those trajectories as the filtered DNS solutions and consider those to be the ground truth. For more details, see [38]. Each trajectory is initiated with a random initial condition. To ensure that the flow is fully developed and has reached the stationary distribution, we discard the first $t = 25$ seconds of the trajectories. Furthermore, for the training of the models we use every 100th state from the high-fidelity simulations in time. Hence, the models take significantly larger time steps than the high-fidelity simulations. We train on 250 time steps and predict up to 750 steps, starting from the same time step, which corresponds to training on the time interval $t \in [25, 37.5]$ and predicting on the time interval $t \in [25, 62.5]$. The choice of temporal resolution is a trade-off between one-step accuracy and rollout stability: a larger physical step size increases the error of the individual one-step prediction but reduces the total number of autoregressive steps required to reach the final horizon T. Conversely, a smaller step size simplifies the one-step mapping but increases the risk of error accumulation through more frequent rollouts. To characterize the dynamics within our chosen regime, we show the temporal autocorrelation of the state in Fig. D.9. We see that after approximately 50 time steps the mean autocorrelation has dropped below 0.25 after which it stays close to zero. This rapid decay demonstrates that our model must handle significant dynamical changes per step, validating the difficulty of the chosen time horizon despite the reduced number of total rollout steps. In general, the trade-off between physical time step size and number of pseudo steps per physical time steps needs careful consideration, and an important suggestion for future work.

[5] https://github.com/agdestein/IncompressibleNavierStokes.jl

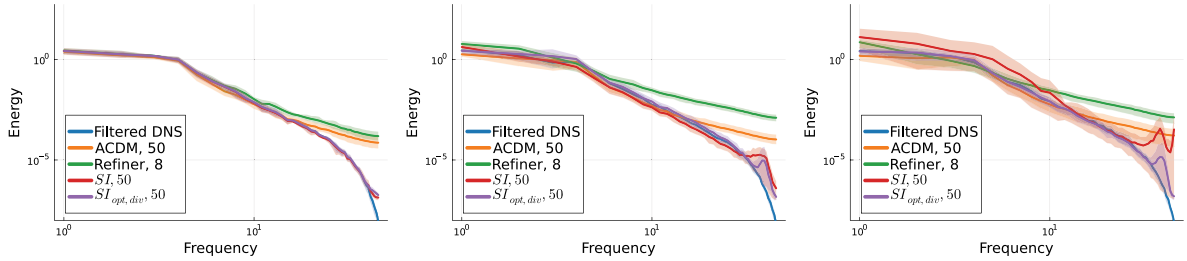
Table 1
Overview of Kolmogorov test case and parameter settings.

Re	10^3
Forcing	Yes
# Train trajectories	45
# Test trajectories	5
# Train time steps	250
# Test time steps	750
# High-fidelity DOFs	$2048^2 \cdot 2 = 8388608$
# Reduced DOFs	$128^2 \cdot 2 = 32768$
High-fidelity step size	$5 \cdot 10^{-4}$
Generative model step size	$100 \cdot 5 \cdot 10^{-4} = 5 \cdot 10^{-2}$
Boundary conditions	Periodic

Note that this problem is similar to that considered in [5], where the authors examine a stochastic variant of the Navier–Stokes equations in vorticity formulation. However, they consider significantly fewer rollouts although they have longer physical time steps. Variants of this test case are also examined in other works, such as [70], which employs a different Reynolds number and physical time step size of 0.002 s with up to 9999 time steps resulting in prediction horizons up to 20.0 s.

We perform several tests using various settings of the models. We train three ACDM models with 10, 25, and 50 diffusion steps, and three Refiner models with 2, 4, and 8 diffusion steps. These choices were made based on the recommendations in the respective papers and for the purpose of comparison with the SI framework. For the SI framework, we train and compare two models. One without the optimized interpolant, and one with the optimized interpolant. The non-optimized interpolant uses $\alpha_\tau = 1 - \tau$ and $\beta_\tau = \tau^2$ as recommended in [5]. Additionally, we also trained an SI model with the linear interpolant, $\alpha_\tau = 1 - \tau$ and $\beta_\tau = \tau$, referred to as SI_{lin}. However, this model appeared to be unstable and the generated trajectories immediately diverged. This corresponds with the empirical findings in [5]. A theoretical study of this phenomenon, as well as more practical studies involving more test cases, are subject to future work. Results with the linear interpolant model are shown in Appendix D. For the optimized interpolants we found that $N_\alpha = N_\beta = 5$ was sufficient to achieve the desired energy distribution properties of the interpolant. We will refer to the optimized SI model as SI_{opt} and the non-optimized as SI. Furthermore, we test SI framework with divergence projection and without. As this is not imposed until after training, no additional models need to be trained. We refer to the models with divergence-free projection as SI_{div} and $\text{SI}_{\text{opt,div}}$.

To simplify the presentation of results, we only show the best results from each model class (ACDM, Refiner, SI, $\text{SI}_{\text{opt,div}}$). We refer to Appendix D for additional results. Furthermore, in Appendix D we also visualize the vorticity and showcase rollouts of up to 10000 physical time steps with the $\text{SI}_{\text{opt,div}}$ model, emphasizing the long stability of the model.



(a) After $n = 10$ time steps. (b) After $n = 100$ time steps. (c) After $n = 750$ time steps.

Fig. 7. Energy spectra for various generative models at three different physical time steps.

Table 2
Results for Kolmogorov. The arrow next to the metric denotes whether larger is better (↑) or smaller is better (↓). Note that we show results for MSE and LSiM averaged over 50 and 750 times steps. Since the Kolmogorov flow is highly chaotic, the long-term performance with respect to those metrics are not representative for the model performance alone. The number following the model is the amount of diffusion/SDE steps in the generation procedure. we highlight the best results in boldface. In cases where there is no significant difference between several results, we highlight all values that are approximately similar.

	Energy W-1 ↓	LSiM ↓		MSE ↓		Corr > 0.8 time ↑
		50 steps	750 steps	50 steps	750 steps	
ACDM, 10	>100	0.52	0.18	>100	>100	0.02
ACDM, 25	16.87	0.18	0.15	0.23	**1.32**	0.39
ACDM, 50	13.14	0.19	0.15	0.20	**1.38**	0.41
Refiner, 2	>100	0.35	0.17	0.40	17.3	0.33
Refiner, 4	>100	0.36	0.20	4.94	>100	0.27
Refiner, 8	21.30	0.20	0.17	0.44	2.38	0.31
SI, 10	80.17	0.13	0.18	0.13	3.51	0.48
SI, 25	44.46	0.14	0.17	0.14	2.59	0.48
SI, 50	37.87	0.13	0.17	0.13	2.44	0.48
$SI_{opt,div}$, 10	4.87	0.09	0.15	0.04	1.56	0.70
$SI_{opt,div}$, 25	**2.69**	**0.06**	0.15	**0.02**	1.638	**0.82**
$SI_{opt,div}$, 50	**2.60**	**0.05**	0.15	**0.02**	1.65	**0.84**

In Fig. 6(a) we see that the alternative methods either over- or undershoot the energy. Furthermore, in Fig. 6(b) we clearly see that the distribution for the $SI_{opt,div}$ method matches the filtered DNS energy significantly better. In Fig. 7, this is further emphasized as we see that the energy spectra are much better matched for both the low and high frequencies. Despite the better performance of the $SI_{opt,div}$ model, we still see a small bump in the high frequencies after a series of physical time steps. This suggests that high-frequency errors accumulate slightly with time. However, the difference between 100 and 750 time steps is small which supports the claim of stable long rollouts. In Fig. D.15 we also see that with more SDE pseudo-time steps this problem becomes smaller.

In Fig. 8 we see the velocity magnitude for various methods at 6 different physical time steps. Qualitatively, we see some differences between the various models. The ACDM and the SI seem generate slightly smoothed states. On the other hand, the Refiner results in states with significantly larger magnitudes. The $SI_{opt,div}$, however, generates trajectories that visually look physically plausible when comparing with the characteristics of the filtered DNS states. This is backed by the results in Figs. 6 and 7.

We summarize the results in Table 2. Here we see that $SI_{opt,div}$ model also performs quantitatively better than the alternatives. We see that the $SI_{opt,div}$ model approximates the energy distribution at least an order of magnitude better than all the alternatives. In particular, the $SI_{opt,div}$ model with only 10 SDE steps outperforms the rest. Furthermore, we see that the $SI_{opt,div}$ method achieves better LSiM accuracy for the first 50 time steps by an order of magnitude. For 750 steps, however, we see that all methods achieve similar accuracy. Since the system is chaotic, this is to be expected for long roll-outs. We see similar behavior for the MSE. Lastly, we see that the $SI_{opt,div}$ method remains correlated with the filtered DNS solution for longer time than the other approaches.

6. Conclusion

In this work, we introduced a novel stochastic generative model for turbulence simulation, leveraging stochastic interpolants to enable probabilistic forecasting while maintaining physical consistency. Unlike conventional generative models, which often fail to incorporate physical constraints, our approach ensures energy-stable time stepping and divergence-free velocity fields, thereby improving both numerical stability and physical reliability. In particular, we have tuned the parameters of the stochastic interpolant in such a way that it is conserving kinetic energy, which is a crucial property in the incompressible Navier–Stokes equations. By training the interpolant on single time steps, we do not need unrolling over multiple time steps.

We demonstrate the effectiveness of our framework on Kolmogorov flow, where it outperforms state-of-the-art generative models, including autoregressive conditional diffusion models (ACDMs) and PDE-Refiners, in terms of energy conservation, spectral accuracy, and long-term stability. Our model not only achieves more accurate statistical properties but also allows for flexible inference, overcoming the rigid step-size constraints of the ACDM and PDE-refiner.

Overall, our findings suggest that stochastic interpolants provide a promising foundation for physics-aware generative modeling in fluid dynamics. Two important limitations are: (i) our framework needs knowledge of k_τ (the average change of energy of the system), which in this article could be set to zero; (ii) we embed energy conservation as a soft constraint through parameterizing the interpolant, and not in a strong way (e.g. through parameterizing the SDE or the NN). Future work will focus on applying the framework to more complex cases, including cases that do not necessarily reach a statistically stationary distribution. Furthermore, we will extend the framework to be able to handle to other relevant physical properties such as entropy, momentum, and symmetries.

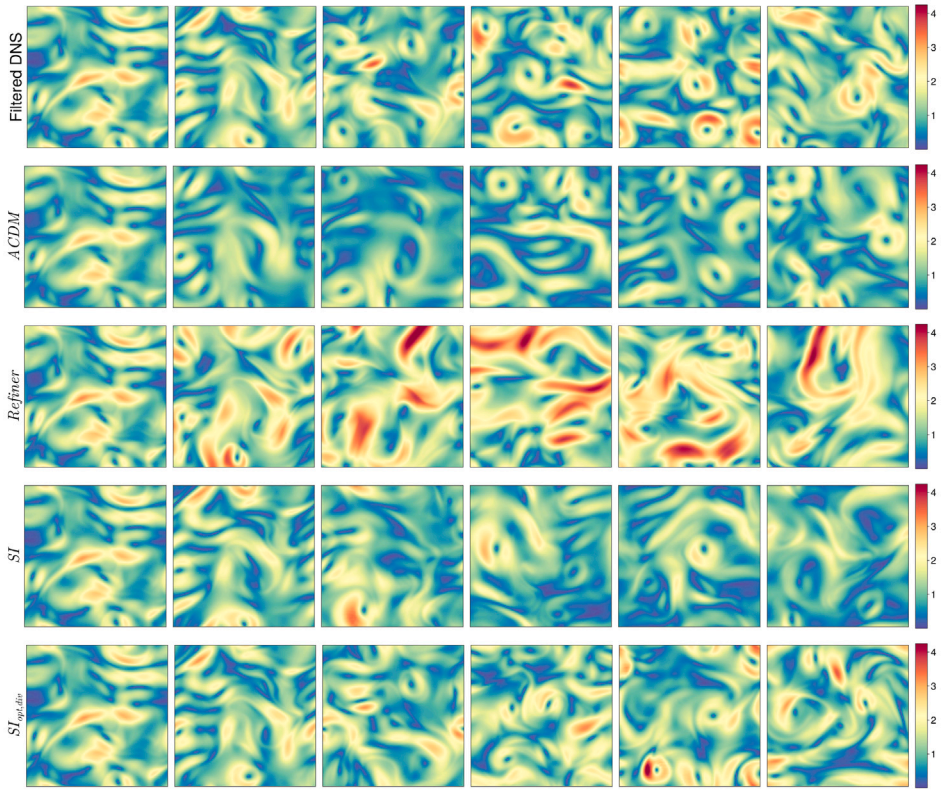


Fig. 8. Velocity magnitude for the various models at different time steps. The same initial condition is used for all realizations. From left to right: $n = 10$, $n = 50$, $n = 100$, $n = 200$, $n = 400$, $n = 750$.

CRediT authorship contribution statement

Nikolaj T. Mücke: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Benjamin Sanderse:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used the Claude large language model inside the Cursor IDE to assist in writing code. After using this tool, the authors reviewed and edited the content as needed and takes full responsibility for the content of the published article.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Benjamin Sanderse reports financial support was provided by National Growth Fund of the Netherlands. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research was funded by the National Growth Fund of the Netherlands and administered by the Netherlands Organization for Scientific Research (NWO) under the AINed XS grant NGF.1609.242.037. The authors furthermore acknowledge the help and support of Syver Agdestein with the Julia package IncompressibleNavierStokes.jl.

Appendix A. Energy conservation and divergence-freeness for the filtered incompressible Navier–Stokes equations

A.1. Kinetic energy conservation

The Navier–Stokes equations, (1)–(2) describe conservation of mass and momentum of a fluid. In the incompressible case, conservation of kinetic energy is a consequence of conservation of mass and momentum, and not a separate conservation law. Conservation of kinetic energy has been used for example to construct stable discretization schemes for turbulent flows [71,72] and stable reduced order models [37].

The kinetic energy is naturally defined as $E := \frac{1}{2}\|\boldsymbol{u}\|_2^2$, where the L_2 norm is given by $\|\boldsymbol{u}\|_2^2 := \langle \boldsymbol{u}, \boldsymbol{u} \rangle$, which is induced by the standard inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle := \int_\Omega \boldsymbol{u} \cdot \boldsymbol{v} \, \mathrm{d}\Omega$. An equation for the evolution of E is derived by differentiating E in time and substituting the momentum equation:

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \frac{\mathrm{d}\frac{1}{2}\langle \boldsymbol{u}, \boldsymbol{u} \rangle}{\mathrm{d}t} = -\langle C(\boldsymbol{u}, \boldsymbol{u}), \boldsymbol{u} \rangle - \langle \nabla p, \boldsymbol{u} \rangle + \langle D\boldsymbol{u}, \boldsymbol{u} \rangle + \langle \boldsymbol{f}(\boldsymbol{u}), \boldsymbol{u} \rangle, \tag{A.1}$$

where we introduced the convection and diffusion operators $C(\boldsymbol{u}, \boldsymbol{u}) := \nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u})$ and $D\boldsymbol{u} := \frac{1}{\mathrm{Re}} \nabla \cdot (\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T)$. The equation simplifies due to three symmetry properties. These symmetry properties will be guiding in designing an energy-consistent SDE. First, due to the fact that $C(\boldsymbol{u}, \boldsymbol{u})$ can be written in a skew-symmetric form (using divergence-freeness), we have $\langle C(\boldsymbol{u}, \boldsymbol{u}), \boldsymbol{u} \rangle = 0$ for periodic or no-slip boundary conditions. Second, the pressure gradient contribution disappears because $\langle \nabla p, \boldsymbol{u} \rangle = \langle p, \nabla \cdot \boldsymbol{u} \rangle = 0$, again using divergence-freeness. Third, due to the symmetry of the diffusive operator we can write $\langle D(\boldsymbol{u}, \boldsymbol{u}), \boldsymbol{u} \rangle = -\langle \nabla \boldsymbol{u}, \nabla \boldsymbol{u} \rangle$. The kinetic energy balance then reduces to

$$\frac{\mathrm{d}E}{\mathrm{d}t} = -\frac{1}{\mathrm{Re}} \|\nabla \boldsymbol{u}\|_2^2 + \langle \boldsymbol{f}(\boldsymbol{u}), \boldsymbol{u} \rangle. \tag{A.2}$$

Consequently, in the absence of boundaries and body forces $\boldsymbol{f}$, the kinetic energy of the flow can only decrease in time, and in inviscid flow it is exactly conserved. The divergence-freeness of the flow field is key in deriving this result. In presence of body forces, like in the Kolmogorov flow from Section 5.1, the dissipation term $\frac{1}{\mathrm{Re}}\|\nabla \boldsymbol{u}\|_2^2$ on average balances the work done by the body force $\langle \boldsymbol{f}(\boldsymbol{u}), \boldsymbol{u}\rangle$.

A.2. Filtering the Navier–Stokes equations

Upon filtering the Navier–Stokes equations with a convolutional filter, the velocity field stays divergence-free:

$$\nabla \cdot \bar{\boldsymbol{u}} = 0, \tag{A.3}$$

because the filter and the divergence operator commute. For the discretized Navier–Stokes equations and a discrete filter, this is in general not true, as the discrete divergence operator and discrete filter do not commute. In [38] we developed a so-called *face-averaging filter* which is such that (A.3) also holds in a discrete sense (provided that $\boldsymbol{u}_h$ is divergence-free):

$$M_h \bar{\boldsymbol{u}}_h = 0, \tag{A.4}$$

where M_h is a matrix representing the discretized divergence operator (on the coarse grid), and $\bar{\boldsymbol{u}}_h := A\boldsymbol{u}_h$. In this way, the discrete filter and discrete divergence operator still commute.

For the momentum equations, filtering does not commute with the nonlinear terms, and the filtered equations feature a so-called commutator error $C(\boldsymbol{u}, \bar{\boldsymbol{u}})$:

$$\frac{\partial \bar{\boldsymbol{u}}}{\partial t} + \nabla \cdot (\bar{\boldsymbol{u}} \otimes \bar{\boldsymbol{u}}) = -\nabla \bar{p} + \frac{1}{\mathrm{Re}} \nabla^2 \bar{\boldsymbol{u}} + \boldsymbol{f}(\bar{\boldsymbol{u}}) + C(\boldsymbol{u}, \bar{\boldsymbol{u}}), \tag{A.5}$$

where $C(\boldsymbol{u}, \bar{\boldsymbol{u}}) = \nabla \cdot (\bar{\boldsymbol{u}} \otimes \bar{\boldsymbol{u}}) - \overline{\nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u})} + \overline{\boldsymbol{f}(\boldsymbol{u})} - \boldsymbol{f}(\bar{\boldsymbol{u}})$. As a consequence, the energy balance is affected, and instead of Eq. (A.2) we have the following evolution for the kinetic energy $\bar{E} := \frac{1}{2}\|\bar{\boldsymbol{u}}\|_2^2$ of the filtered field:

$$\frac{\mathrm{d}\bar{E}}{\mathrm{d}t} = -\frac{1}{\mathrm{Re}}\|\nabla \bar{\boldsymbol{u}}\|_2^2 + \langle \boldsymbol{f}(\bar{\boldsymbol{u}}), \bar{\boldsymbol{u}}\rangle + \langle C(\boldsymbol{u}, \bar{\boldsymbol{u}}), \bar{\boldsymbol{u}}\rangle. \tag{A.6}$$

Note that we have omitted the explicit dependence on the grid size in the norm and the inner product. Including the grid size only adds a scaling factor and does not change the outcome of the derivations. In the absence of body forces, viscosity and boundary contributions, $\bar{E}$ is not a conserved quantity (in contrast to E), due to the additional term $C(\boldsymbol{u}, \bar{\boldsymbol{u}})$ which can be both positive and negative, and accounts for exchange of energy with unresolved scales. In statistically stationary flow, the terms on the right hand side balance each other on average.

During time-stepping such as with the explicit schemes in [73,74], first a tentative velocity field is computed as a solution to the momentum equations, and then a projection is performed to make this velocity field divergence-free. This projection of any non-divergence-free field $\bar{\boldsymbol{u}}_h^*$ can be written as

$$\bar{\boldsymbol{u}}_h = \Pi \bar{\boldsymbol{u}}_h^*, \tag{A.7}$$

where $\Pi = I - M_h^T L_h^{-1} M_h$, and $L_h = M_h M_h^T$ is a Poisson matrix. In practice, Eq. (A.7) is solved in a two-step process:

$$L_h \phi = M_h \bar{\boldsymbol{u}}_h^*, \tag{A.8}$$

$$\bar{\boldsymbol{u}}_h = \bar{\boldsymbol{u}}_h^* - M_h^T \phi. \tag{A.9}$$

Appendix B. Proof of Theorem 3.1

Here, we provide a proof of Theorem 3.1.

Proof. To ease the notation, we omit the explicit dependence on $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ and write $\boldsymbol{I}_\tau := \boldsymbol{I}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1)$ and $\boldsymbol{R}_\tau := \boldsymbol{R}_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1)$, where $(\boldsymbol{x}_0, \boldsymbol{x}_1) \sim p(\boldsymbol{x}_0, \boldsymbol{x}_1)$.

We remind the reader that the dynamics of the interpolant is governed by the SDE:

$$\mathrm{d}\boldsymbol{I}_\tau = \boldsymbol{R}_\tau \mathrm{d}\tau + \gamma_\tau \mathrm{d}\boldsymbol{W}_\tau, \quad \tau \in [0,1], \quad \boldsymbol{X}_0 = \boldsymbol{x}_0, \tag{B.1}$$

where

$$\boldsymbol{R}_\tau = \dot{\alpha}_\tau \boldsymbol{x}_0 + \dot{\beta}_\tau \boldsymbol{x}_1 + \dot{\gamma}_\tau \boldsymbol{W}_\tau \tag{B.2}$$

For any quantity of interest $Q(\boldsymbol{I}_\tau)$, we can define a process, $Y_\tau = Q(\boldsymbol{I}_\tau)$. The time evolution of Y_τ is given by Itô's lemma (Theorem 7.3.1 in [67]):

$$\begin{aligned} \mathrm{d}Y_\tau &= \left[\frac{\partial Q}{\partial \tau}\mathrm{d}\tau + \nabla_{\boldsymbol{I}} Q \cdot \boldsymbol{R}_\tau + \frac{1}{2}\gamma_\tau^2 (\nabla_{\boldsymbol{I}} \nabla_{\boldsymbol{I}}^T Q)\right] \mathrm{d}\tau + \gamma_\tau \nabla_{\boldsymbol{I}} Q \cdot \mathrm{d}\boldsymbol{W}_\tau \\ &= \left[\nabla_{\boldsymbol{I}} Q \cdot \boldsymbol{R}_\tau + \frac{1}{2}\gamma_\tau^2 (\nabla_{\boldsymbol{I}} \nabla_{\boldsymbol{I}}^T Q)\right] \mathrm{d}\tau + \gamma_\tau \nabla_{\boldsymbol{I}} Q \cdot \mathrm{d}\boldsymbol{W}_\tau. \end{aligned} \tag{B.3}$$

Note that $\frac{\partial Q}{\partial \tau} = 0$ as Q is not explicitly dependent on τ. By setting the quantity of interest to be the kinetic energy, $Q(I_\tau) = \frac{1}{2}\|\boldsymbol{I}_\tau\|_2^2$, and using

$$\nabla_X \|X\|_2^2 = 2X, \quad \nabla_X \nabla_X^T \|X\|_2^2 = 2d, \tag{B.4}$$

for any $X \in \mathbb{R}^d$, we get the first result

$$\mathrm{d}Y_\tau = \left[\boldsymbol{I}_\tau \cdot \boldsymbol{R}_\tau + \frac{d}{2}\gamma_\tau^2\right] \mathrm{d}\tau + \gamma_\tau \boldsymbol{I}_\tau \cdot \mathrm{d}\boldsymbol{W}_\tau. \tag{B.5}$$

For the second result, the expression for the expected energy evolution, we start by expanding the dot product, $\boldsymbol{I}_\tau \cdot \boldsymbol{R}_\tau$:

$$\begin{aligned} \boldsymbol{I}_\tau \cdot \boldsymbol{R}_\tau =& \dot{\alpha}_\tau \alpha_\tau \|\boldsymbol{x}_0\|_2^2 + \dot{\beta}_\tau \beta_\tau \|\boldsymbol{x}_1\|_2^2 + \dot{\gamma}_\tau \gamma_\tau \|\boldsymbol{W}_\tau\|_2^2 \\ &+ (\dot{\beta}_\tau \alpha_\tau + \dot{\alpha}_\tau \beta_\tau)\langle \boldsymbol{x}_0, \boldsymbol{x}_1\rangle + \alpha_\tau \dot{\gamma}_\tau \langle \boldsymbol{x}_0, \boldsymbol{W}_\tau\rangle + \beta_\tau \dot{\gamma}_\tau \langle \boldsymbol{x}_1, \boldsymbol{W}_\tau\rangle. \end{aligned} \tag{B.6}$$

Taking the expected value with respect to $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, and $\boldsymbol{W}_\tau$ gives:

$$\begin{aligned} \mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{W}_\tau)}\left[\boldsymbol{I}_\tau \cdot \boldsymbol{R}_\tau\right] = \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{x}_1}\Big[&\dot{\alpha}_\tau \alpha_\tau \|\boldsymbol{x}_0\|_2^2 \\ &+ \dot{\beta}_\tau \beta_\tau \|\boldsymbol{x}_1\|_2^2 + (\dot{\beta}_\tau \alpha_\tau + \dot{\alpha}_\tau \beta_\tau)\langle \boldsymbol{x}_0, \boldsymbol{x}_1\rangle\Big] + \dot{\gamma}_\tau \gamma_\tau \tau d. \end{aligned} \tag{B.7}$$

Here, we used that $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ are independent with respect to $\boldsymbol{W}_\tau$ and that $\mathbb{E}[\|\boldsymbol{W}_\tau\|_2^2] = \tau d$. Lastly, by taking the expected value of $\mathrm{d}Y_\tau$ and using that $\mathbb{E}\left[\boldsymbol{I}_\tau \cdot \mathrm{d}\boldsymbol{W}_\tau\right] = 0$, we get:

$$\begin{aligned} &\mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{W})}\left[\mathrm{d}Y_\tau\right] \\ &= \left(\mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_1)}\left[\dot{\alpha}_\tau \alpha_\tau \|\boldsymbol{x}_0\|_2^2 + \dot{\beta}_\tau \beta_\tau \|\boldsymbol{x}_1\|_2^2 + (\dot{\beta}_\tau \alpha_\tau + \dot{\alpha}_\tau \beta_\tau)\langle \boldsymbol{x}_0, \boldsymbol{x}_1\rangle\right]\right. \\ &\quad \left. + \dot{\gamma}_\tau \gamma_\tau \tau d + \frac{d}{2}\gamma_\tau^2\right) \mathrm{d}\tau. \end{aligned} \tag{B.8}$$

It does not make a difference whether we include the last two terms in the expected value in Eq. (B.8), as they are independent from $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$. Hence, by including the last two terms in the expected value, we get:

$$\mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{W})}\left[\mathrm{d}Y_\tau\right] = \mathbb{E}_{(\boldsymbol{x}_0, \boldsymbol{x}_1)}\left[H_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1; \alpha_\tau, \beta_\tau, \gamma_\tau)\right] \mathrm{d}\tau, \tag{B.9}$$

with

$$\begin{aligned} &H_\tau(\boldsymbol{x}_0, \boldsymbol{x}_1; \alpha_\tau, \beta_\tau, \gamma_\tau) \\ &= \dot{\alpha}_\tau \alpha_\tau \|\boldsymbol{x}_0\|_2^2 + \dot{\beta}_\tau \beta_\tau \|\boldsymbol{x}_1\|_2^2 + (\dot{\beta}_\tau \alpha_\tau + \dot{\alpha}_\tau \beta_\tau)\langle \boldsymbol{x}_0, \boldsymbol{x}_1\rangle + \dot{\gamma}_\tau \gamma_\tau \tau d + \frac{d}{2}\gamma_\tau^2, \end{aligned} \tag{B.10}$$

which was what we wanted to show. □

Appendix C. Metrics and quantities of interest

Mean squared error. The mean squared error (MSE) is computed as:

$$\mathrm{MSE}(\bar{\boldsymbol{u}}_h, \bar{\boldsymbol{v}}_h) = \frac{1}{N_T N_x} \sum_{n=1}^{N_T} \left\|\bar{\boldsymbol{u}}_h^n - \bar{\boldsymbol{v}}_h^n\right\|_2^2, \tag{C.1}$$

where N_T is the number of physical time steps and $\|\cdot\|_2^2$ is the squared l^2-norm. We compute the MSE between each generated trajectory and a filtered DNS trajectory simulated with the same initial condition and

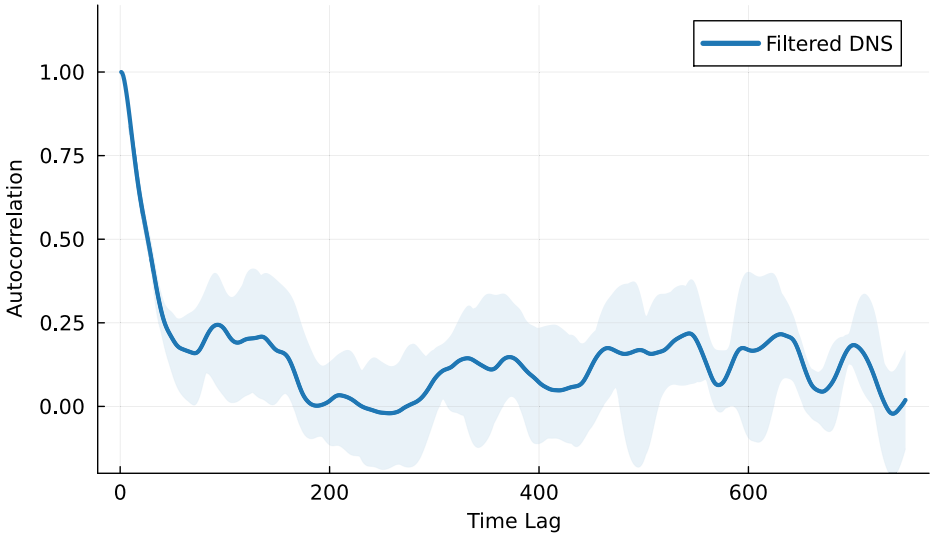


Fig. D.9. Temporal autocorrelation for 5 filtered DNS trajectories. The line is the mean autocorrelation and the shaded area shows the standard deviation.

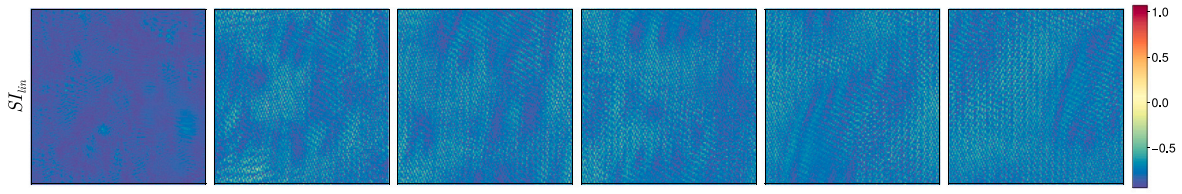


Fig. D.10. Velocity magnitude at different time steps for the stochastic interpolant model with linear coefficients. The same initial condition and color bar range as in Fig. 8 are used. Time steps from left to right: $n = 10$, $n = 50$, $n = 100$, $n = 200$, $n = 400$, $n = 750$.

compute the average over all computed MSEs. Note that the MSE is not a useful metric for long-term predictions when dealing with chaotic systems.

Kinetic energy. The kinetic energy for at a time step n is computed by:

$$E(\bar{\boldsymbol{u}}_h^n) = \frac{1}{2h^2} \left\| \bar{\boldsymbol{u}}_h^n \right\|_2^2, \tag{C.2}$$

where h is the equidistant spatial grid size in x- and y- direction. We are especially interested in assessing whether the generative models generate data that follows the same kinetic energy distribution as the filtered DNS trajectories.

Rate of change. The rate of change (RoC) at time step n is computed as in [2] by:

$$RoC(\bar{\boldsymbol{u}}_h^n) = \left\| (\bar{\boldsymbol{u}}_h^n - \bar{\boldsymbol{u}}_h^{n-1})/\Delta t \right\|_1, \tag{C.3}$$

where Δt is the physical step size. The RoC measures whether a trajectory follows the expected evolution. For example, if a trajectory explodes, the RoC grows substantially and if the RoC goes to zero the trajectory goes to a steady state. This metric is particularly useful for long roll-outs where the predictions are not expected to follow the true state exactly, but are expected to follow the general evolution. This is especially relevant when dealing with chaotic trajectories.

Wasserstein-1 distance. To assess the quality of the approximations of the energy distributions, we compute the Wasserstein-1 (W-1) distance. This is computed by:

$$W(p,q) = \inf_{\gamma \in \Gamma(p,q)} \mathbb{E}_{(x,y)\sim\gamma(x,y)} \left[\|x - y\|_1 \right] \tag{C.4}$$

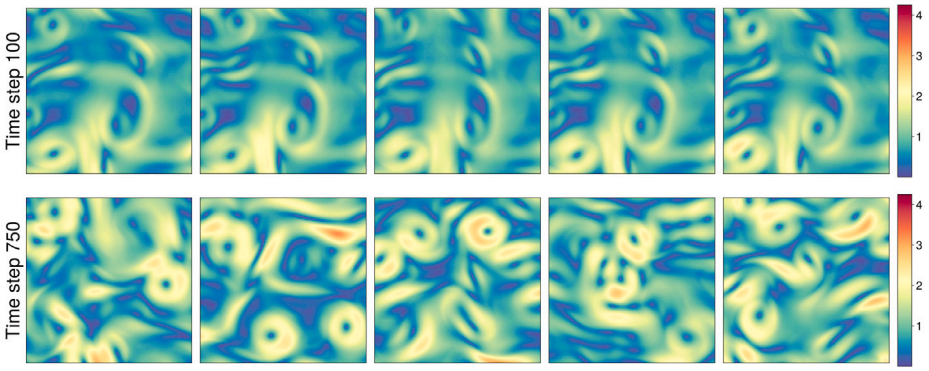
where $\Gamma(p,q)$ is the set of all couplings between the distributions p and q and $\|\cdot\|_1$ is the l^1 norm. We compute the Wasserstein metric between the total kinetic energy of the filtered DNS simulations and the generated simulations. For a trajectory, the total kinetic energy is computed at each time-step. Then, the collection of energies from each time step is used to make the empirical distribution.

LSiM. LSiM is a similarity metric designed for numerical simulations. The metric is computed by encoding the data with a trained neural network and then comparing the latent representations. For details, see [68]. We compute the LSiM between each trajectory in the ensemble of generated states and the filtered DNS state at each time step. Then, the mean and standard deviation over all time steps and trajectories are reported. For a state with multiple fields, such as velocity in x-direction and y-direction, we consider each channel separately and report the mean. For a field, we convert the values into an RGB representation, as the LSiM is designed to handle RGB images. This is done following the same procedure as in [68].

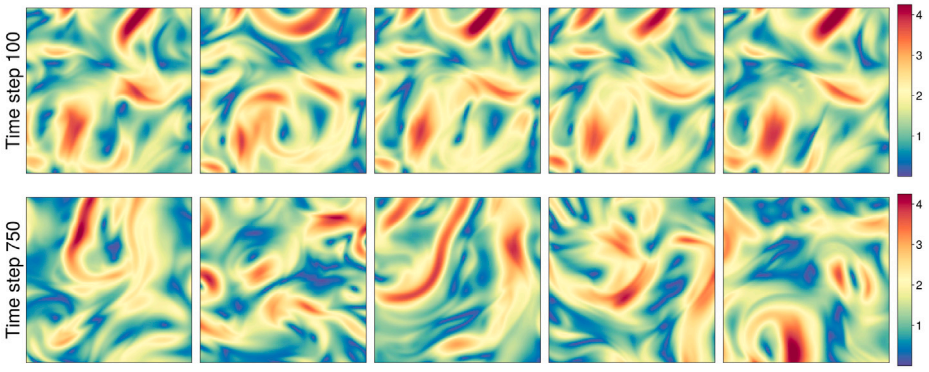
Pearson correlation. We compute the Pearson correlation coefficient between the generated state and the filtered DNS state at each time step. In the beginning the correlation is approximately 1, but will deteriorate with time. We report the time it takes for the correlation to drop below 0.8.

Appendix D. Additional results - Kolmogorov flow

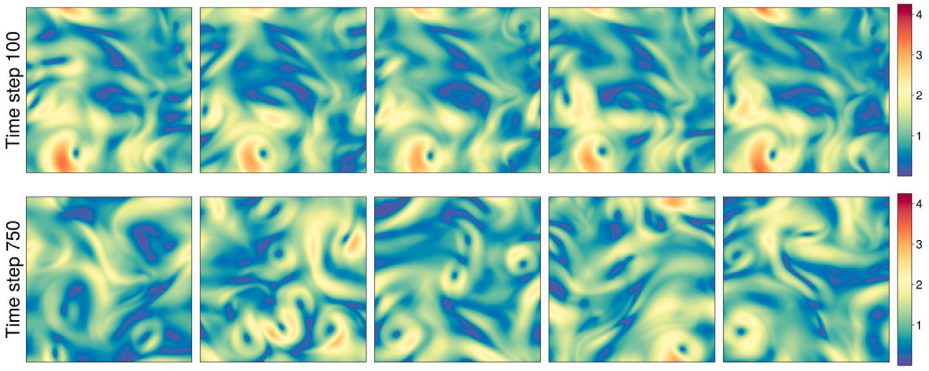
See Figs. D.9–D.19.



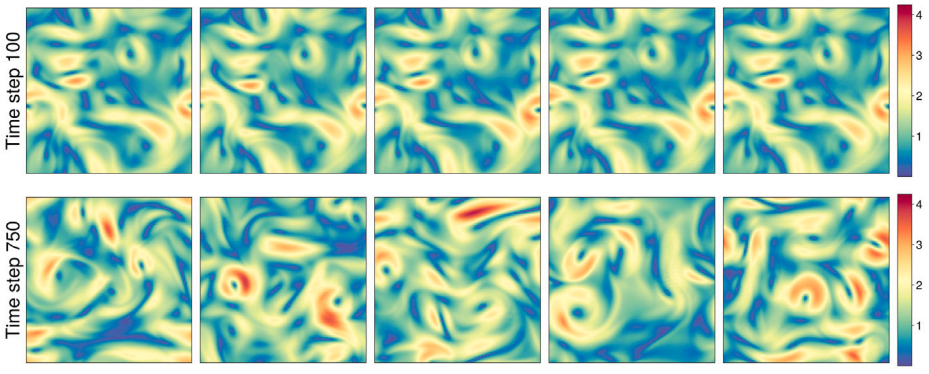
(a) ACDM with 50 pseudo-steps.



(b) Refiner with 8 pseudo-steps.

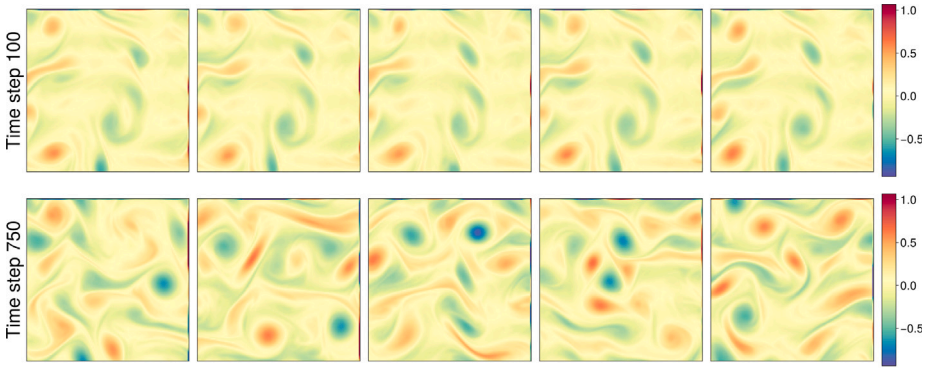


(c) Stochastic inerpolant with 50 pseudo-steps.

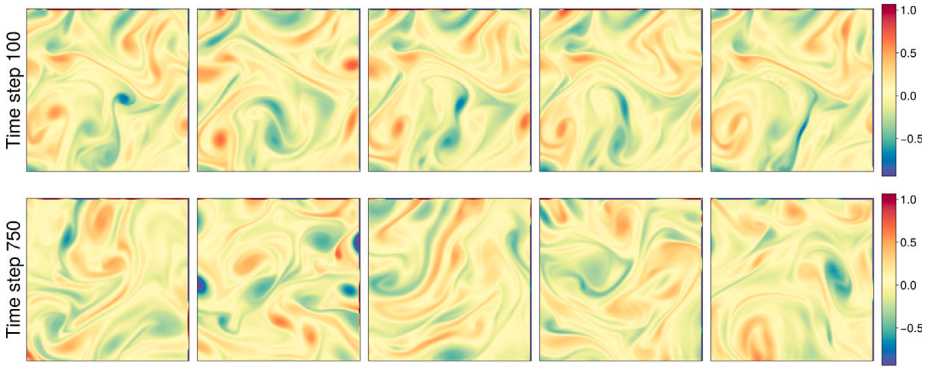


(d) Optimized stochastic interpolant with divergence project and 50 pseudo-steps.

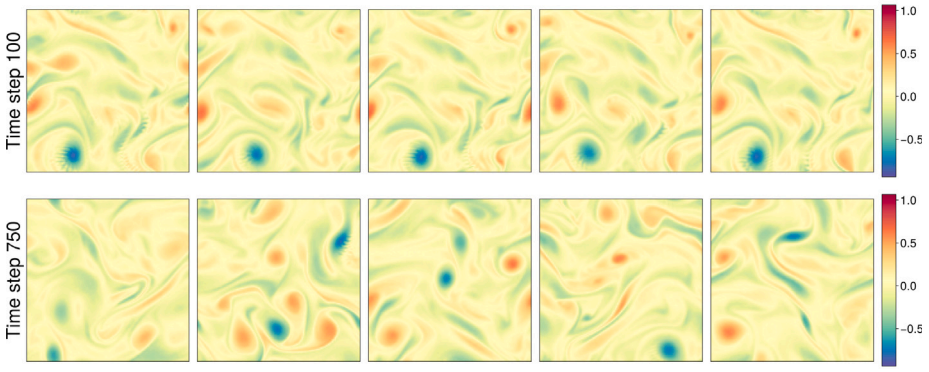
Fig. D.11. Five velocity magnitude realizations at time step 100 and 750 for various models.



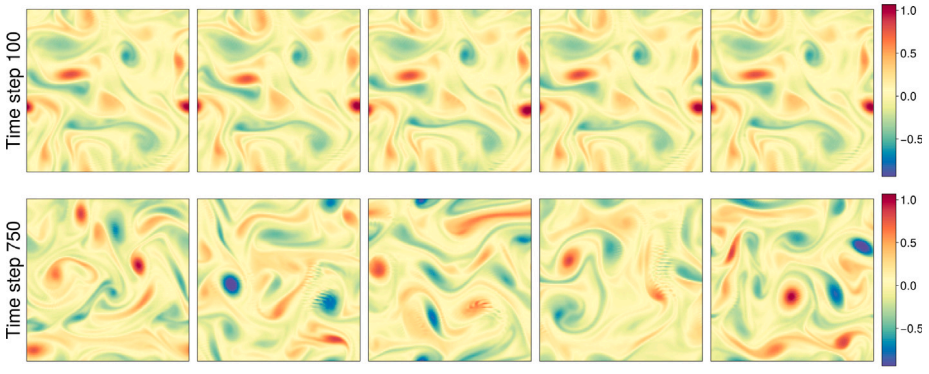
(a) ACDM with 50 pseudo-steps.



(b) Refiner with 8 pseudo-steps.



(c) Stochastic inerpolant with 50 pseudo-steps.



(d) Optimized stochastic interpolant with divergence project and 50 pseudo-steps.

Fig. D.12. Five vorticity realizations at time step 100 and 750 for various models.

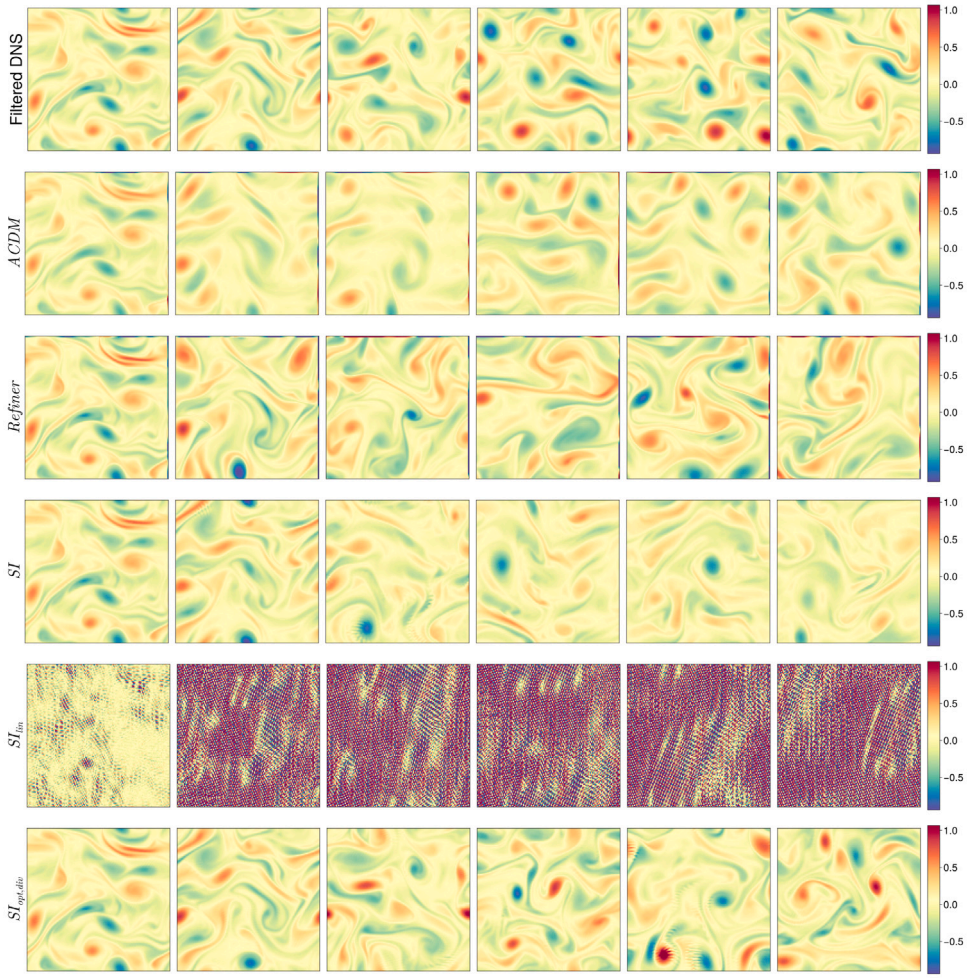


Fig. D.13. Vorticity for the various models at different time steps. The same initial condition is used for all realizations. From left to right: $n = 10$, $n = 50$, $n = 100$, $n = 200$, $n = 400$, $n = 750$.

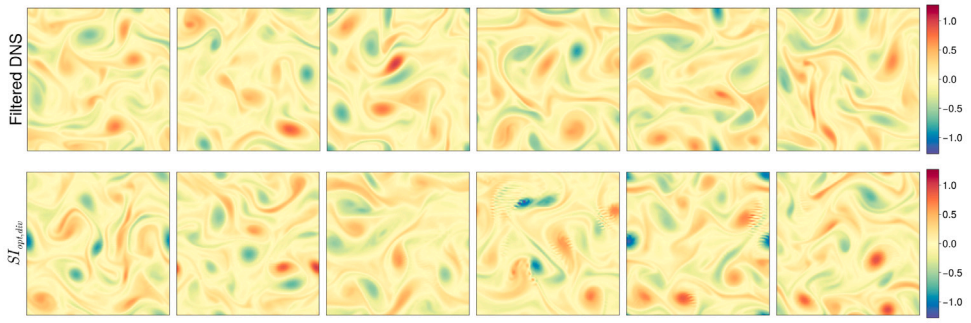
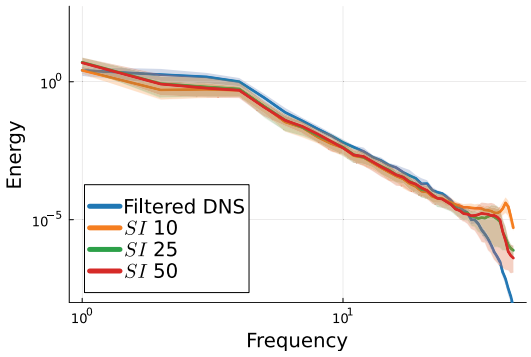
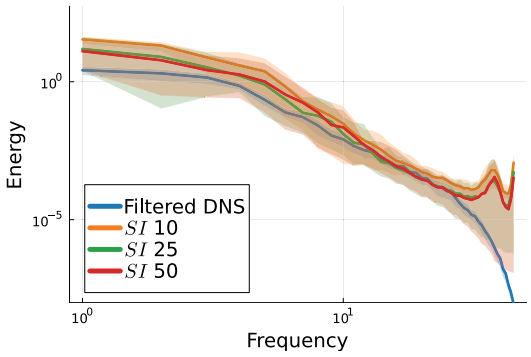


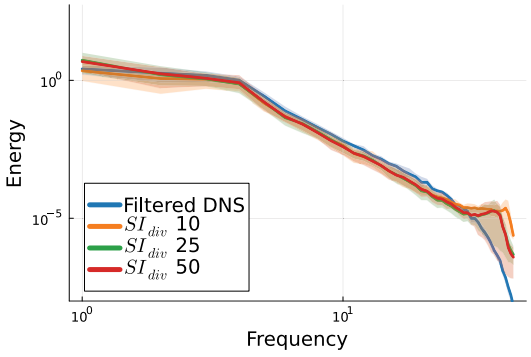
Fig. D.14. Vorticity for a high-fidelity trajectory and a trajectory generated with the optimized and divergence-free SI model at different time steps. The same initial condition is used for all realizations. From left to right: $n = 500$, $n = 1000$, $n = 2500$, $n = 5000$, $n = 7500$, $n = 10000$, corresponding to physical times $t = 25$ s, $t = 50$ s, $t = 125$ s, $t = 250$ s, $t = 375$ s, and $t = 500$ s.



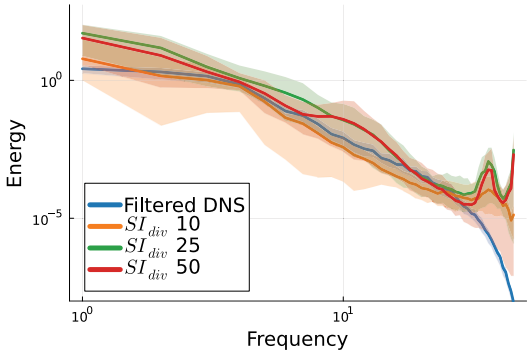
(a) Stochastic interpolant, n=200



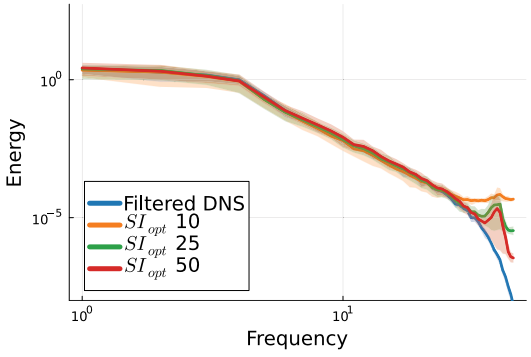
(b) Stochastic interpolant, n=750



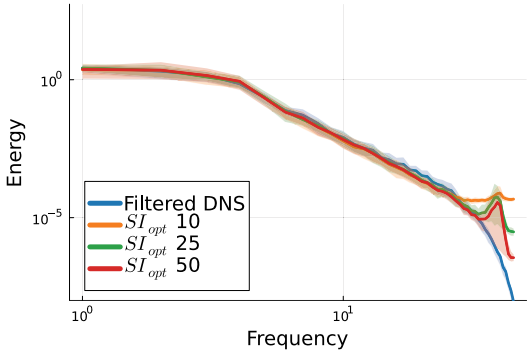
(c) Stochastic interpolant with divergence-free projection, n=200



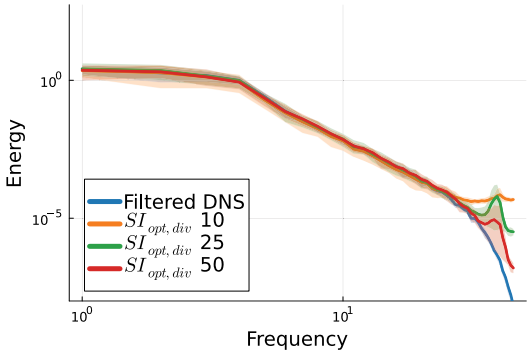
(d) Stochastic interpolant with divergence-free projection, n=750



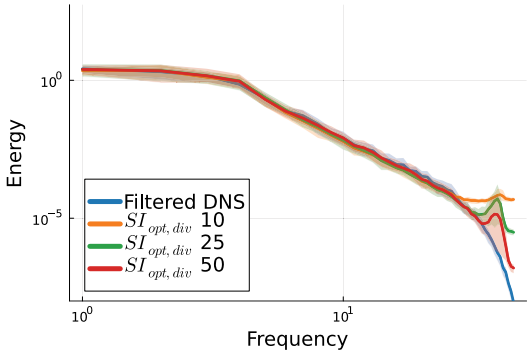
(e) Optimized stochastic interpolant, n=200



(f) Optimized stochastic interpolant, n=750

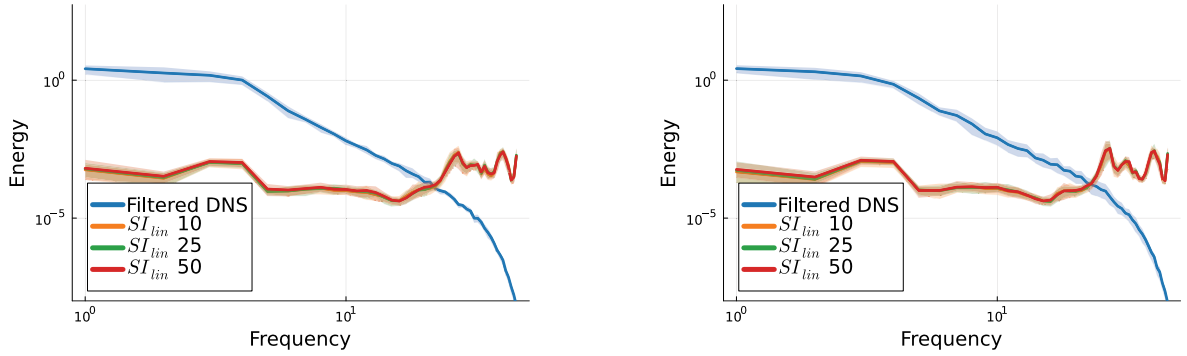


(g) Optimized stochastic interpolant with divergence-free projection, n=200



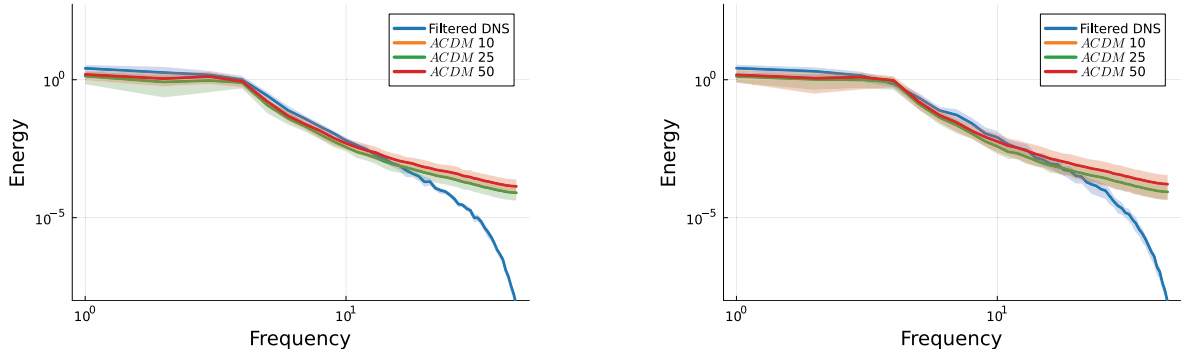
(h) Optimized stochastic interpolant with divergence-free projection, n=750

Fig. D.15. Energy spectra for the stochastic interpolants.



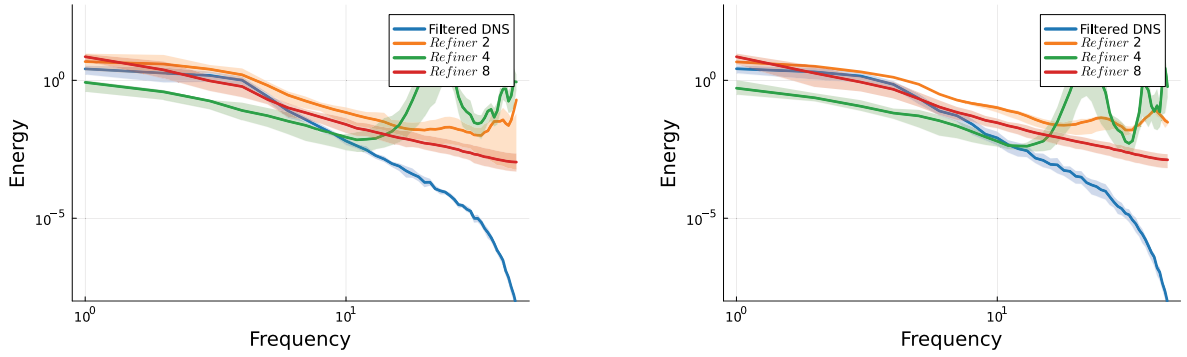
(a) Stochastic interpolant with linear interpolant, n=200

(b) Stochastic interpolant with linear interpolant, n=750



(c) ACDM, n=200

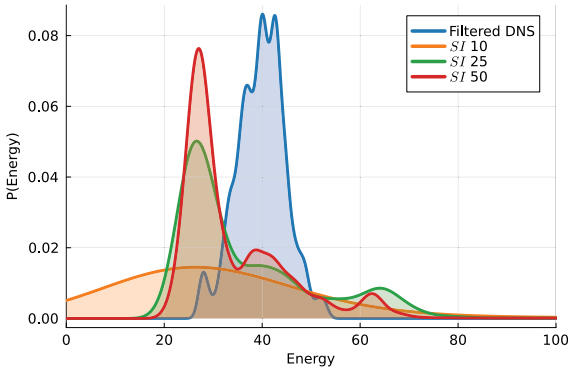
(d) ACDM, n=750



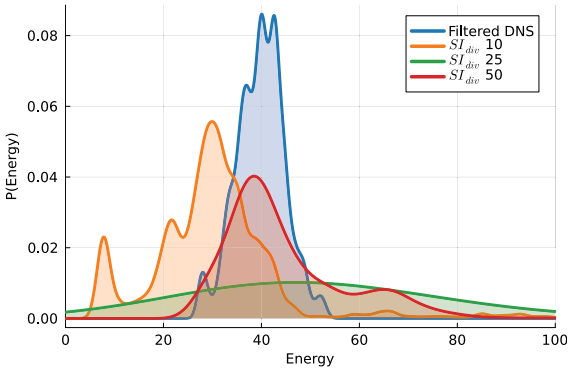
(e) Refiner, n=200

(f) Refiner, n=750

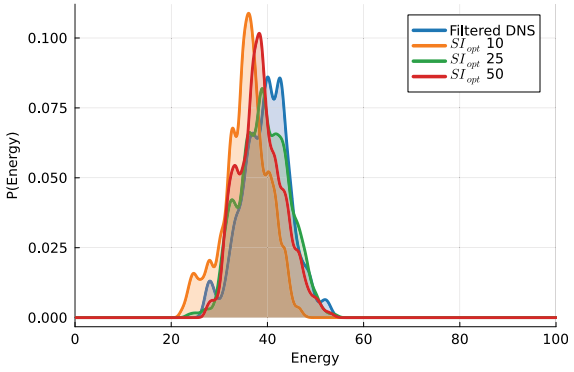
Fig. D.16. Energy spectra for the linear stochastic interpolant, ACDM, and PDE-Refiner.



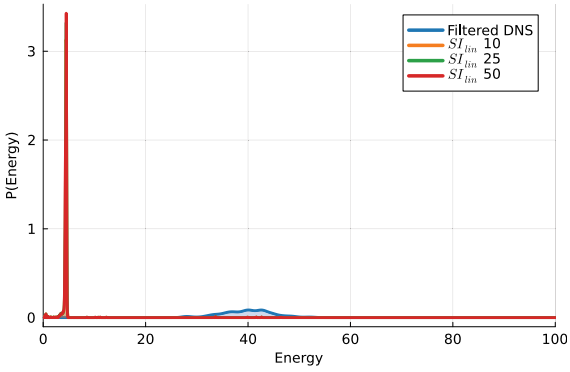
(a) Stochastic interpolant



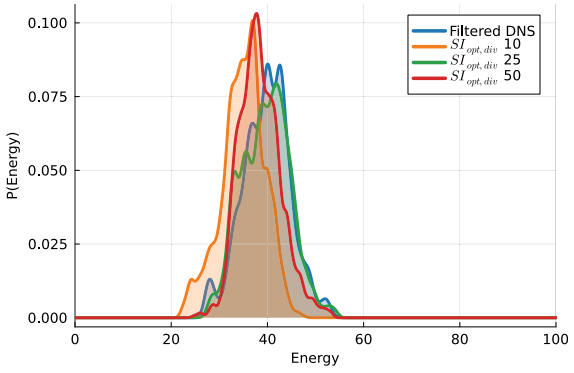
(b) Stochastic interpolant with divergence-free projection



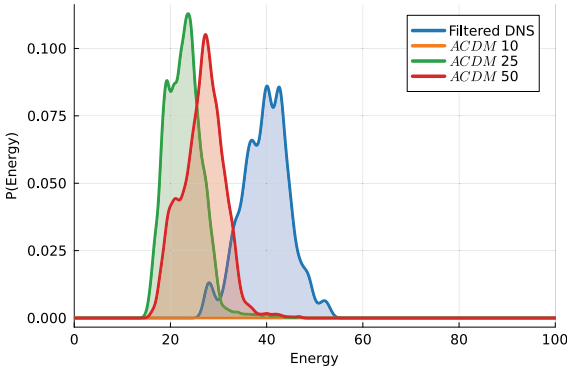
(c) Optimized stochastic interpolant



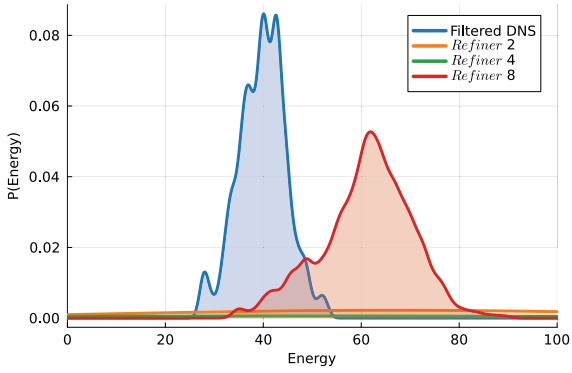
(d) Stochastic interpolant with linear interpolant.



(e) Optimized stochastic interpolant with divergence-free projection

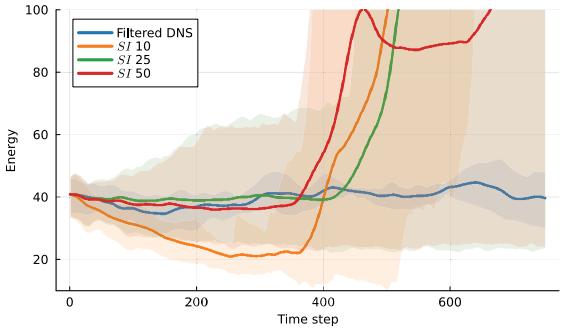


(f) ACDM

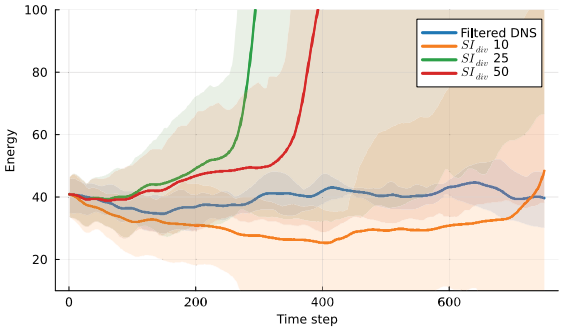


(g) PDE-Refiner

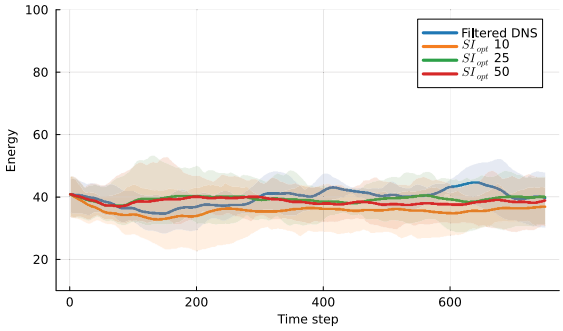
Fig. D.17. Probability density function of the energy. Note that the distributions for the ACDM with 10 steps and the PDE-refiner are not in the figures, since they are centered far away from the true distribution.



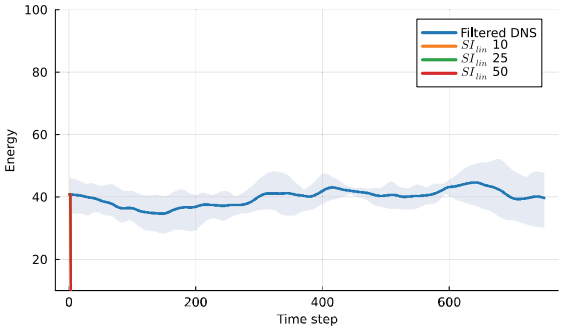
(a) Stochastic interpolant



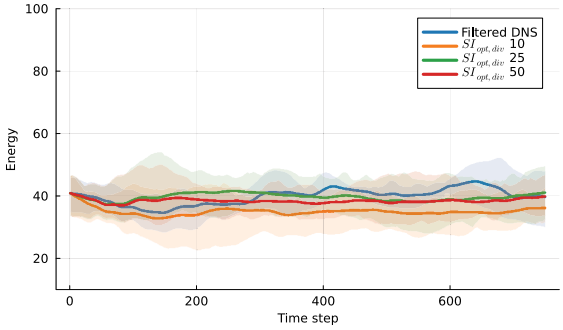
(b) Stochastic interpolant with divergence-free projection



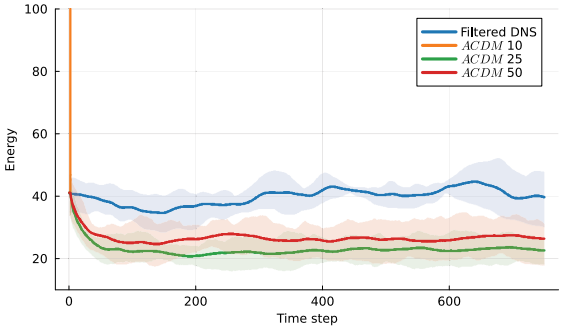
(c) Optimized stochastic interpolant



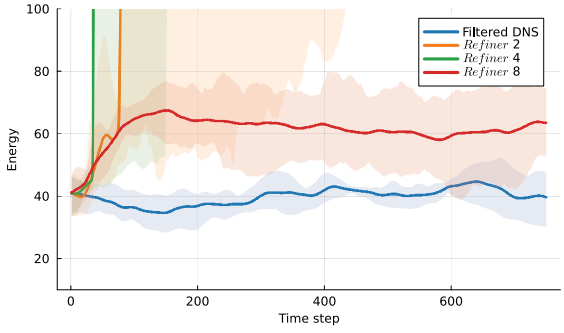
(d) Stochastic interpolant with linear interpolant.



(e) Optimized stochastic interpolant with divergence-free projection

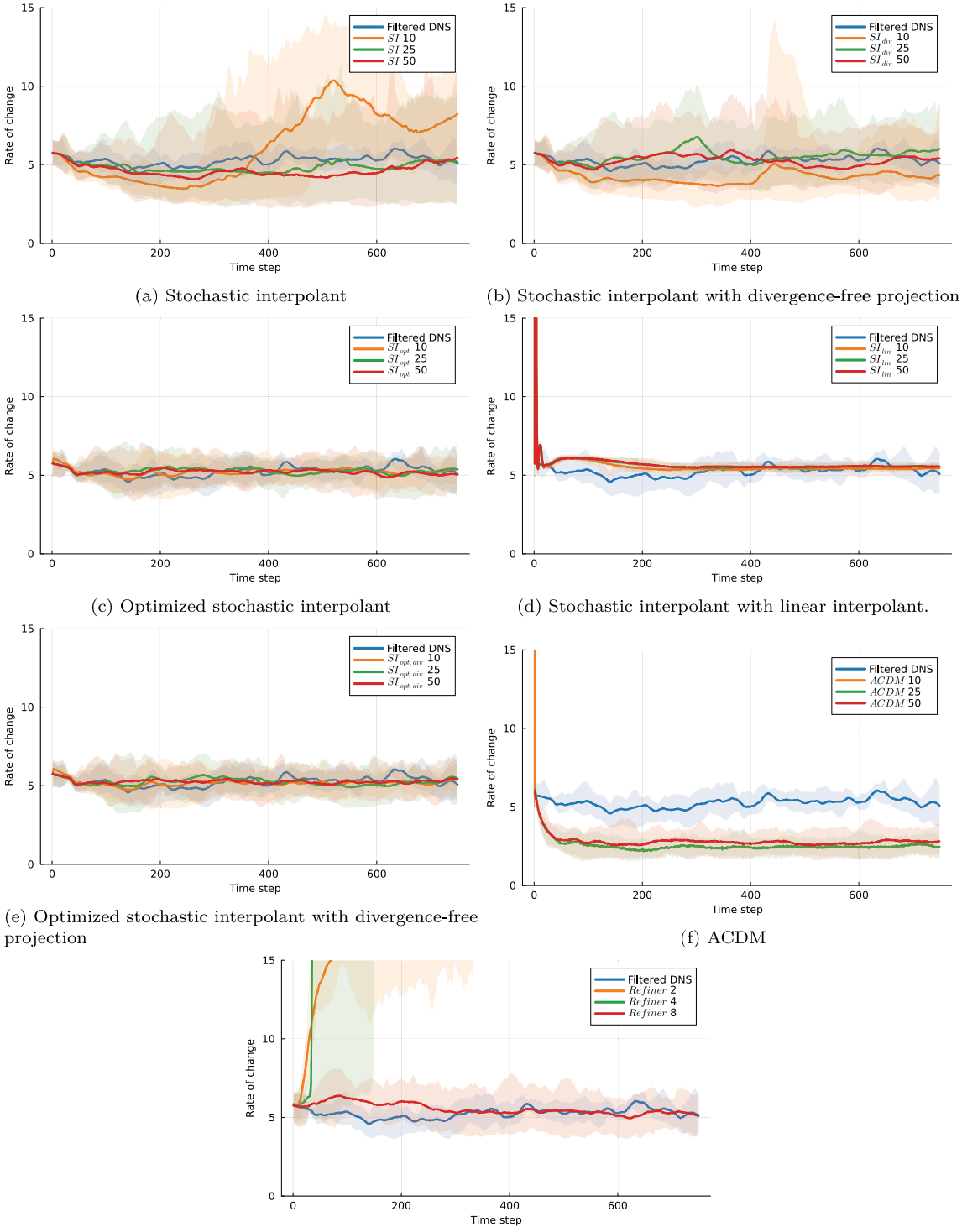


(f) ACDM



(g) PDE-refiner

Fig. D.18. Energy evolution.



(a) Stochastic interpolant

(b) Stochastic interpolant with divergence-free projection

(c) Optimized stochastic interpolant

(d) Stochastic interpolant with linear interpolant.

(e) Optimized stochastic interpolant with divergence-free projection

(f) ACDM

(g) PDE-refiner

Fig. D.19. Rate of change.

References

[1] Boral A, Wan ZY, Zepeda-Núñez L, Lottes J, Wang Q, Chen Y-f, Anderson J, Sha F. Neural ideal large eddy simulation: Modeling turbulence with neural stochastic differential equations. Adv Neural Inf Process Syst 2023;36:69270–83.

[2] Kohl G, Chen L-W, Thuerey N. Benchmarking autoregressive conditional diffusion models for turbulent flow simulation. 2024, http://dx.doi.org/10.48550/arXiv.2309.01745, arXiv:2309.01745.

[3] Price I, Sanchez-Gonzalez A, Alet F, Andersson TR, El-Kadi A, Masters D, Ewalds T, Stott J, Mohamed S, Battaglia P, Lam R, Willson M. Probabilistic weather forecasting with machine learning. Nature 2025;637(8044):84–90. http://dx.doi.org/10.1038/s41586-024-08252-9, URL https://www.nature.com/articles/s41586-024-08252-9.

[4] Shehata Y, Holzschuh B, Thuerey N. Improved sampling of diffusion models in fluid dynamics with tweedie's formula. In: The thirteenth international conference on learning representations. 2025.

[5] Chen Y, Goldstein M, Hua M, Albergo MS, Boffi NM, Vanden-Eijnden E. Probabilistic Forecasting with Stochastic Interpolants and Föllmer Processes. In:

Proceedings of the 41st international conference on machine learning. 2024, URL https://proceedings.mlr.press/v235/chen24n.html.

[6] Chen N. Stochastic Methods for Modeling and Predicting Complex Dynamical Systems: Uncertainty Quantification, State Estimation, and Reduced-Order Models. Synthesis lectures on mathematics & statistics, Cham: Springer International Publishing; 2023, http://dx.doi.org/10.1007/978-3-031-22249-8.

[7] Hesthaven JS, Ubbiali S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. J Comput Phys 2018;363:55–78. http://dx.doi.org/10.1016/j.jcp.2018.02.037, URL https://www.sciencedirect.com/science/article/pii/S0021999118301190.

[8] Mücke NT, Bohté SM, Oosterlee CW. Reduced order modeling for parameterized time-dependent PDEs using spatially and memory aware deep learning. J Comput Sci 2021;53:101408. http://dx.doi.org/10.1016/j.jocs.2021.101408, URL https://www.sciencedirect.com/science/article/pii/S1877750321000934.

[9] Geneva N, Zabaras N. Transformers for modeling physical systems. Neural Netw 2022;146:272–89. http://dx.doi.org/10.1016/j.neunet.2021.11.022, URL https://www.sciencedirect.com/science/article/pii/S0893608021004500.

[10] Bodnar C, Bruinsma WP, Lucic A, Stanley M, Allen A, Brandstetter J, Garvan P, Riechert M, Weyn JA, Dong H, Gupta JK, Thambiratnam K, Archibald AT, Wu C-C, Heider E, Welling M, Turner RE, Perdikaris P. A foundation model for the Earth system. Nature 2025;641(8065):1180–7. http://dx.doi.org/10.1038/s41586-025-09005-y, URL https://www.nature.com/articles/s41586-025-09005-y.

[11] Pathak J, Subramanian S, Harrington P, Raja S, Chattopadhyay A, Mardani M, Kurth T, Hall D, Li Z, Azizzadenesheli K, Hassanzadeh P, Kashinath K, Anandkumar A. FourCastNet: A global data-driven high-resolution weather model using adaptive Fourier neural operators. 2022, http://dx.doi.org/10.48550/arXiv.2202.11214, arXiv:2202.11214.

[12] Nguyen T, Brandstetter J, Kapoor A, Gupta JK, Grover A. ClimaX: A foundation model for weather and climate. 2023, http://dx.doi.org/10.48550/arXiv.2301.10343, arXiv:2301.10343.

[13] Lam R, Sanchez-Gonzalez A, Willson M, Wirnsberger P, Fortunato M, Alet F, Ravuri S, Ewalds T, Eaton-Rosen Z, Hu W, Merose A, Hoyer S, Holland G, Vinyals O, Stott J, Pritzel A, Mohamed S, Battaglia P. Learning skillful medium-range global weather forecasting. Science 2023. http://dx.doi.org/10.1126/science.adi2336, URL https://www.science.org/doi/10.1126/science.adi2336.

[14] Bommasani R, Hudson DA, Adeli E, Altman R, Arora S, von Arx S, Bernstein MS, Bohg J, Bosselut A, Brunskill E, Brynjolfsson E, Buch S, Card D, Castellon R, Chatterji N, Chen A, Creel K, Davis JQ, Demszky D, Donahue C, Doumbouya M, Durmus E, Ermon S, Etchemendy J, Ethayarajh K, Fei-Fei L, Finn C, Gale T, Gillespie L, Goel K, Goodman N, Grossman S, Guha N, Hashimoto T, Henderson P, Hewitt J, Ho DE, Hong J, Hsu K, Huang J, Icard T, Jain S, Jurafsky D, Kalluri P, Karamcheti S, Keeling G, Khani F, Khattab O, Koh PW, Krass M, Krishna R, Kuditipudi R, Kumar A, Ladhak F, Lee M, Lee T, Leskovec J, Levent I, Li XL, Li X, Ma T, Malik A, Manning CD, Mirchandani S, Mitchell E, Munyikwa Z, Nair S, Narayan A, Narayanan D, Newman B, Nie A, Niebles JC, Nilforoshan H, Nyarko J, Ogut G, Orr L, Papadimitriou I, Park JS, Piech C, Portelance E, Potts C, Raghunathan A, Reich R, Ren H, Rong F, Roohani Y, Ruiz C, Ryan J, Ré C, Sadigh D, Sagawa S, Santhanam K, Shih A, Srinivasan K, Tamkin A, Taori R, Thomas AW, Tramèr F, Wang RE, Wang W, Wu B, Wu J, Wu Y, Xie SM, Yasunaga M, You J, Zaharia M, Zhang M, Zhang T, Zhang X, Zhang Y, Zheng L, Zhou K, Liang P. On the Opportunities and Risks of Foundation Models. 2022, http://dx.doi.org/10.48550/arXiv.2108.07258, arXiv:2108.07258.

[15] Batatia I, Benner P, Chiang Y, Elena AM, Kovács DP, Riebesell J, Advincula XR, Asta M, Avaylon M, Baldwin WJ, Berger F, Bernstein N, Bhowmik A, Blau SM, Cărare V, Darby JP, De S, Pia FD, Deringer VL, Elijošius R, El-Machachi Z, Falcioni F, Fako E, Ferrari AC, Genreith-Schriever A, George J, Goodall REA, Grey CP, Grigorev P, Han S, Handley W, Heenen HH, Hermansson K, Holm C, Jaafar J, Hofmann S, Jakob KS, Jung H, Kapil V, Kaplan AD, Karimitari N, Kermode JR, Kroupa N, Kullgren J, Kuner MC, Kuryla D, Liepuoniute G, Margraf JT, Magdău I-B, Michaelides A, Moore JH, Naik AA, Niblett SP, Norwood SW, O'Neill N, Ortner C, Persson KA, Reuter K, Rosen AS, Schaaf LL, Schran C, Shi BX, Sivonxay E, Stenczel TK, Svahn V, Sutton C, Swinburne TD, Tilly J, van der Oord C, Varga-Umbrich E, Vegge T, Vondrák M, Wang Y, Witt WC, Zills F, Csányi G. A foundation model for atomistic materials chemistry. 2024, http://dx.doi.org/10.48550/arXiv.2401.00096, arXiv:2401.00096.

[16] Rosen Y, Roohani Y, Agarwal A, Samotorčan L, Consortium TS, Quake SR, Leskovec J. Universal cell embeddings: A foundation model for cell biology. 2023, http://dx.doi.org/10.1101/2023.11.28.568918, 2023.11.28.568918.

[17] Alkin B, Fürst A, Schmid S, Gruber L, Holzleitner M, Brandstetter J. Universal physics transformers: A framework for efficiently scaling neural operators. Adv Neural Inf Process Syst 2024;37:25152–94, URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/2cd36d327f33d47b372d4711edd08de0-Abstract-Conference.html.

[18] Hao Z, Su C, Liu S, Berner J, Ying C, Su H, Anandkumar A, Song J, Zhu J. DPOT: auto-regressive denoising operator transformer for large-scale PDE pre-training. In: Proceedings of the 41st international conference on machine learning. JMLR.org; 2024.

[19] McCabe M, Régaldo-Saint Blancard B, Parker L, Ohana R, Cranmer M, Bietti A, Eickenberg M, Golkar S, Krawezik G, Lanusse F, et al. Multiple physics pretraining for spatiotemporal surrogate models. Adv Neural Inf Process Syst 2024;37:119301–35.

[20] Shen J, Marwah T, Talwalkar A. UPS: Efficiently building foundation models for PDE solving via cross-modal adaptation. Trans Mach Learn Res 2024. URL https://openreview.net/forum?id=0r9mhjRv1E.

[21] Subramanian S, Harrington P, Keutzer K, Bhimji W, Morozov D, Mahoney MW, Gholami A. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. In: Thirty-seventh conference on neural information processing systems. 2023, URL https://openreview.net/forum?id=zANxvzflMl.

[22] Yang L, Liu S, Meng T, Osher SJ. In-context operator learning with data prompts for differential equation problems. Proc Natl Acad Sci 2023;120(39):e2310142120. http://dx.doi.org/10.1073/pnas.2310142120, arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.2310142120, URL https://www.pnas.org/doi/abs/10.1073/pnas.2310142120.

[23] Herde M, Raonic B, Rohner T, Käppeli R, Molinaro R, de Bézenac E, Mishra S. Poseidon: Efficient foundation models for pdes. Adv Neural Inf Process Syst 2024;37:72525–624.

[24] Keisler R. Forecasting Global Weather with Graph Neural Networks. 2022, http://dx.doi.org/10.48550/arXiv.2202.07575, arXiv:2202.07575.

[25] Li L, Carver R, Lopez-Gomez I, Sha F, Anderson J. Generative emulation of weather forecast ensembles with diffusion models. Sci Adv 2024;10(13):eadk4489. http://dx.doi.org/10.1126/sciadv.adk4489.

[26] Kochkov D, Yuval J, Langmore I, Norgaard P, Smith J, Mooers G, Klöwer M, Lottes J, Rasp S, Düben P, Hatfield S, Battaglia P, Sanchez-Gonzalez A, Willson M, Brenner MP, Hoyer S. Neural general circulation models for weather and climate. Nature 2024;632(8027):1060–6. http://dx.doi.org/10.1038/s41586-024-07744-y.

[27] Pope SB. Ten questions concerning the large-eddy simulation of turbulent flows. New J Phys 2004;6(1):35. http://dx.doi.org/10.1088/1367-2630/6/1/035.

[28] Berner J, Achatz U, Batté L, Bengtsson L, de la Cámara A, Christensen HM, Colangeli M, Coleman DRB, Crommelin D, Dolaptchiev SI, Franzke CLE, Friederichs P, Imkeller P, Järvinen H, Juricke S, Kitsios V, Lott F, Lucarini V, Mahajan S, Palmer TN, Penland C, Sakradzija M, von Storch J-S, Weisheimer A, Weniger M, Williams PD, Yano J-I. Stochastic parameterization: Toward a new view of weather and climate models. Bull Am Meteorol Soc 2017;98(3):565–88. http://dx.doi.org/10.1175/BAMS-D-15-00268.1.

[29] Li X, Wong T-KL, Chen RTQ, Duvenaud D. Scalable gradients for stochastic differential equations. In: Proceedings of the twenty third international conference on artificial intelligence and statistics. 2020, URL https://proceedings.mlr.press/v108/li20i.html.

[30] Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models. Adv Neural Inf Process Syst 2020;33:6840–51.

[31] Song Y, Sohl-Dickstein J, Kingma DP, Kumar A, Ermon S, Poole B. Score-based generative modeling through stochastic differential equations. 2021, http://dx.doi.org/10.48550/arXiv.2011.13456, arXiv:2011.13456, [cs] URL http://arxiv.org/abs/2011.13456.

[32] Melnik A, Ljubljanac M, Lu C, Yan Q, Ren W, Ritter H. Video diffusion models: A survey. 2024, http://dx.doi.org/10.48550/arXiv.2405.03150, arXiv:2405.03150, [cs], URL http://arxiv.org/abs/2405.03150.

[33] Li T, Chen M, Guo B, Shen Z. A survey on diffusion language models. 2025, http://dx.doi.org/10.48550/arXiv.2508.10875, arXiv:2508.10875, [cs], URL http://arxiv.org/abs/2508.10875.

[34] Albergo MS, Vanden-Eijnden E. Building normalizing flows with stochastic interpolants. 2023, http://dx.doi.org/10.48550/arXiv.2209.15571, arXiv:2209.15571.

[35] Albergo MS, Boffi NM, Vanden-Eijnden E. Stochastic interpolants: A unifying framework for flows and diffusions. 2023, http://dx.doi.org/10.48550/arXiv.2303.08797, arXiv:2303.08797, [cs], URL http://arxiv.org/abs/2303.08797.

[36] van Gastelen T, Edeling W, Sanderse B. Energy-conserving neural network for turbulence closure modeling. J Comput Phys 2024;508:113003. http://dx.doi.org/10.1016/j.jcp.2024.113003.

[37] Sanderse B. Non-linearly stable reduced-order models for incompressible flow with energy-conserving finite volume methods. J Comput Phys 2020;421:109736. http://dx.doi.org/10.1016/j.jcp.2020.109736.

[38] Agdestein SD, Sanderse B. Discretize first, filter next: Learning divergence-consistent closure models for large-eddy simulation. J Comput Phys 2025;522:113577. http://dx.doi.org/10.1016/j.jcp.2024.113577.

[39] Park J, Choi H. Toward neural-network-based large eddy simulation: Application to turbulent channel flow. J Fluid Mech 2021;914:A16. http://dx.doi.org/10.1017/jfm.2020.931.

[40] Kurz M, Beck A. Investigating model-data inconsistency in data-informed turbulence closure terms. In: 14th WCCM-ECCOMAS congress 2020. 2021, http://dx.doi.org/10.23967/wccm-eccomas.2020.115.

[41] Rasp S. Coupled online learning as a way to tackle instabilities and biases in neural network parameterizations: General algorithms and Lorenz 96 case study (v1.0). Geosci Model Dev 2020;13(5):2185–96. http://dx.doi.org/10.5194/gmd-13-2185-2020.

[42] Sanderse B, Stinis P, Maulik R, Ahmed SE. Scientific machine learning for closure models in multiscale problems: A review. Found Data Sci 2024;7(1):298–337. http://dx.doi.org/10.3934/fods.2024043.
[43] Sagaut P. Large eddy simulation for incompressible flows: An introduction, In: Scientific computation, 3rd ed. Berlin ; New York: Springer; 2006.
[44] Pope SB. Turbulent flows. Cambridge University Press; 2000.
[45] Ahmed SE, Pawar S, San O, Rasheed A, Iliescu T, Noack BR. On closures for reduced order models—A spectrum of first-principle to machine-learned avenues. Phys Fluids 2021;33(9):091301. http://dx.doi.org/10.1063/5.0061577.
[46] Kohl G, Chen L-W, Thuerey N. Benchmarking autoregressive conditional diffusion models for turbulent flow simulation. 2024, http://dx.doi.org/10.48550/arXiv.2309.01745, arXiv:2309.01745, [cs] version: 2, URL http://arxiv.org/abs/2309.01745.
[47] Dong X, Chen C, Wu J-L. Data-driven stochastic closure modeling via conditional diffusion model and neural operator. J Comput Phys 2025;534:114005. http://dx.doi.org/10.1016/j.jcp.2025.114005, URL https://www.sciencedirect.com/science/article/pii/S0021999125002888.
[48] Molinaro R, Lanthaler S, Raonić B, Rohner T, Armegioiu V, Simonis S, Grund D, Ramic Y, Wan ZY, Sha F, Mishra S, Zepeda-Núñez L. Generative AI for fast and accurate statistical computation of fluids. 2025, http://dx.doi.org/10.48550/arXiv.2409.18359, arXiv:2409.18359.
[49] Beck A, Flad D, Munz C-D. Deep neural networks for data-driven LES closure models. J Comput Phys 2019;398:108910. http://dx.doi.org/10.1016/j.jcp.2019.108910, URL https://www.sciencedirect.com/science/article/pii/S0021999119306151.
[50] Melchers H, Crommelin D, Koren B, Menkovski V, Sanderse B. Comparison of neural closure models for discretised PDEs. Comput Math Appl 2023;143:94–107. http://dx.doi.org/10.1016/j.camwa.2023.04.030.
[51] Brantner B, Romemont Gd, Kraus M, Li Z. Volume-preserving transformers for learning time series data with structure. 2024, http://dx.doi.org/10.48550/arXiv.2312.11166, arXiv:2312.11166, [math] URL http://arxiv.org/abs/2312.11166.
[52] Dener A, Miller MA, Churchill RM, Munson T, Chang C-S. Training neural networks under physical constraints using a stochastic augmented Lagrangian approach. 2020, http://dx.doi.org/10.48550/arXiv.2009.07330, arXiv:2009.07330, [physics] URL http://arxiv.org/abs/2009.07330.
[53] Foias C, Manley O, Rosa R, Temam R. Navier-Stokes equations and turbulence. vol. 83, Cambridge University Press; 2001.
[54] Kingma DP, Salimans T, Poole B, Ho J. Variational diffusion models. 2023, http://dx.doi.org/10.48550/arXiv.2107.00630, arXiv:2107.00630, [cs] URL http://arxiv.org/abs/2107.00630.
[55] Benamou J-D, Brenier Y. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. Numer Math 2000;84(3):375–93. http://dx.doi.org/10.1007/s002110050002, URL http://link.springer.com/10.1007/s002110050002.
[56] Liu X, Gong C, Liu Q. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. 2022, URL https://openreview.net/forum?id=XVjTT1nw5z.
[57] Lippe P, Veeling B, Perdikaris P, Turner R, Brandstetter J. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. Adv Neural Inf Process Syst 2023;36:67398–433.
[58] Ronneberger O, Fischer P, Brox T. U-Net: Convolutional networks for biomedical image segmentation. 2015, http://dx.doi.org/10.1007/978-3-319-24574-4_28, URL https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28.
[59] Liu Z, Mao H, Wu C-Y, Feichtenhofer C, Darrell T, Xie S. A ConvNet for the 2020s. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022, p. 11976–86.
[60] Springenberg JT, Dosovitskiy A, Brox T, Riedmiller M. Striving for Simplicity: The All Convolutional Net. 2015, http://dx.doi.org/10.48550/arXiv.1412.6806, arXiv:1412.6806, [cs] URL http://arxiv.org/abs/1412.6806.
[61] Zeiler MD, Krishnan D, Taylor GW, Fergus R. Deconvolutional networks. In: 2010 IEEE computer society conference on computer vision and pattern recognition. 2010, p. 2528–35. http://dx.doi.org/10.1109/CVPR.2010.5539957, URL https://ieeexplore.ieee.org/document/5539957.
[62] Peebles W, Xie S. Scalable diffusion models with transformers. In: Proceedings of the IEEE/CVF international conference on computer vision. 2023, p. 4195–205.
[63] Mücke NT, Bohté SM, Oosterlee CW. The deep latent space particle filter for real-time data assimilation with uncertainty quantification. Sci Rep 2024;14(1):19447. http://dx.doi.org/10.1038/s41598-024-69901-7, URL https://www.nature.com/articles/s41598-024-69901-7.
[64] Hendrycks D, Gimpel K. Gaussian Error Linear Units (GELUs). 2023, http://dx.doi.org/10.48550/arXiv.1606.08415, arXiv:1606.08415, [cs] URL http://arxiv.org/abs/1606.08415.
[65] Loshchilov I, Hutter F. Decoupled Weight Decay Regularization. 2019, http://dx.doi.org/10.48550/arXiv.1711.05101, arXiv:1711.05101, [cs] URL http://arxiv.org/abs/1711.05101.
[66] Loshchilov I, Hutter F. SGDR: Stochastic gradient descent with warm restarts. 2017, http://dx.doi.org/10.48550/arXiv.1608.03983, arXiv:1608.03983, [cs] URL http://arxiv.org/abs/1608.03983.
[67] Thygesen UH. Stochastic differential equations for science and engineering. New York: Chapman and Hall/CRC; 2023, http://dx.doi.org/10.1201/9781003277569.
[68] Kohl G, Um K, Thuerey N. Learning similarity metrics for numerical simulations. In: Proceedings of the 37th international conference on machine learning. 2020, URL https://proceedings.mlr.press/v119/kohl20a.html.
[69] Agdestein SD, Ciarella S, Sanderse B. IncompressibleNavierStokes.jl. 2024, original-date: 2021-09-22T08:15:00Z URL https://github.com/agdestein/IncompressibleNavierStokes.jl.
[70] Pedersen C, Zanna L, Bruna J. Thermalizer: Stable autoregressive neural emulation of spatiotemporal chaos. 2025, arXiv preprint arXiv:2503.18731.
[71] Coppola G, Capuano F, de Luca L. Discrete energy-conservation properties in the numerical simulation of the Navier–Stokes equations. Appl Mech Rev 2019;71(1):010803. http://dx.doi.org/10.1115/1.4042820.
[72] Verstappen R, Veldman A. Symmetry-preserving discretization of turbulent flow. J Comput Phys 2003;187(1):343–68. http://dx.doi.org/10.1016/S0021-9991(03)00126-8.
[73] Agdestein SD, Sanderse B. Discretize first, filter next: Learning divergence-consistent closure models for large-eddy simulation. J Comput Phys 2025;522:113577. http://dx.doi.org/10.1016/j.jcp.2024.113577, URL https://www.sciencedirect.com/science/article/pii/S0021999124008258.
[74] Sanderse B, Koren B. Accuracy analysis of explicit Runge–Kutta methods applied to the incompressible Navier–Stokes equations. J Comput Phys 2012;231(8):3041–63. http://dx.doi.org/10.1016/j.jcp.2011.11.028.