Benchmarking Autoregressive Conditional Diffusion Models for Turbulent Flow Simulation

Georg Kohl [1] **Li-Wei Chen** [1] **Nils Thuerey** [1]

Abstract

Simulating turbulent flows is crucial for a wide range of applications, and machine learning-based solvers are gaining increasing relevance. However, achieving temporal stability when generalizing to longer rollout horizons remains a persistent challenge for learned PDE solvers. In this work, we analyze if fully data-driven fluid solvers that utilize an autoregressive rollout based on conditional diffusion models are a viable option to address this challenge. We investigate accuracy, posterior sampling, spectral behavior, and temporal stability, while requiring that methods generalize to flow parameters beyond the training regime. To quantitatively and qualitatively benchmark the performance of a range of flow prediction approaches, three challenging scenarios including incompressible and transonic flows, as well as isotropic turbulence are employed. We find that even simple diffusion-based approaches can outperform multiple established flow prediction methods in terms of accuracy and temporal stability, while being on par with state-of-the-art stabilization techniques like unrolling at training time. Such traditional architectures are superior in terms of inference speed, however, the probabilistic nature of diffusion approaches allows for inferring multiple predictions that align with the statistics of the underlying physics.

1. Introduction

Simulations based on partial differential equations (PDEs), particularly those involving turbulent fluid flows, constitute a crucial research area with applications ranging from medicine (Olufsen et al., 2000) to climate research (Wyngaard, 1992), as well as numerous engineering fields (Moin & Mahesh, 1998; Verma et al., 2018). Historically, such flows have been simulated via iterative numerical solvers for the Navier-Stokes equations. Recently, there has been a growing interest in combining or replacing traditional solvers with deep learning. However, despite the significant progress made in this field, a major remaining challenge is the ability to predict rollouts that maintain both stability and accuracy over longer temporal horizons (Kochkov et al., 2021; Um et al., 2020). Fluid simulations are inherently complex and dynamic, and therefore, it is highly challenging to accurately capture the intricate physical phenomena that occur over extended periods of time. Additionally, due to their chaotic nature, even small ambiguities of the spatially averaged states used for simulations can lead to fundamentally different solutions over time (Pope, 2000). However, most learned methods and traditional numerical solvers process simulation trajectories deterministically, and thus only provide a single answer.

We explore these issues by investigating the usefulness of the recently emerging conditional diffusion models (Ho et al., 2020; Song et al., 2021b) for turbulent flows, which serve as representatives for more general PDE-based simulations. Specifically, we are interested in the probabilistic prediction of fluid flow trajectories from an initial condition. We aim for answering the question: *Are autoregressive diffusion models competitive tools for modeling fluid simulations compared to other established architectures?* For this purpose, we analyze model accuracy, temporal rollout stability, spectral behavior, posterior sampling capabilities, as well as computational costs for different approaches. The central aims and outcomes of this work are as follows: (***i***) We combine a conditional diffusion approach with an autoregressive rollout to produce a probabilistic surrogate flow simulator, that represents state of the art diffusion models applied to flow prediction. (***ii***) We broadly compare this approach on flow prediction problems with increasing difficulty against a range of common architectures in terms of accuracy, posterior sampling, temporal stability, and statistical correspondence to the underlying physical behavior. (***iii***) We show that even simple, diffusion-based flow predictors are remarkably competitive with state-of-the-art stabilization techniques like unrolling at training time, while providing diverse, physically plausible posterior samples.

[1]Technical University of Munich, Germany. Correspondence to: Georg Kohl <georg.kohl@tum.de>.

AI for Science Workshop at the 41st International Conference on Machine Learning, Vienna, Austria. 2024. Copyright 2024 by the author(s).

2. Related Work

Fluid Solvers utilizing Machine Learning A variety of works have used machine learning as means to improve numerical solvers. Several approaches focus on learning computational stencils (Bar-Sinai et al., 2019; Kochkov et al., 2021) or additive corrections (de Avila Belbute-Peres et al., 2020; List et al., 2022; Um et al., 2020) to increase simulation accuracy. When the solver is not integrated into the computational graph, typically a data-driven surrogate model is trained to replace the solver. Convolutional neural networks (CNNs) for such flow prediction problems are very popular, and often employ an encoder-processor-decoder architecture. For the latent space processor, multilayer perceptrons (Kim et al., 2019) as well as LSTMs (Wiewel et al., 2019) were proposed. As particularly successful latent architectures, transformers (Vaswani et al., 2017) have also been combined with CNN-based encoders as reduced-order models (Hemmasian & Farimani, 2023; Geneva & Zabaras, 2022). Alternatives do not rely on an autoregressive latent model, e.g., by using spatio-temporal 3D convolutions (Deo et al., 2023), dilated convolutions (Stachenfeld et al., 2022), or problem-specific multi-scale architectures (Wang et al., 2020). Furthermore, various works utilize message passing architectures (Brandstetter et al., 2022; Pfaff et al., 2021). Adding noise to training inputs was likewise proposed to improve temporal prediction stability for graph networks (Sanchez-Gonzalez et al., 2020). Transformer-based latent models have also been combined with a graph network encoder and decoder in previous work (Han et al., 2021).

Diffusion Models Diffusion models (Hyvärinen, 2005; Sohl-Dickstein et al., 2015) became popular after diffusion probabilistic models and denoising score matching were combined for high-quality unconditional image generation (Ho et al., 2020). This approach has since been improved in many aspects, e.g., with meaningful latent representations (Song et al., 2021a) or better sampling (Nichol & Dhariwal, 2021). Diffusion models for image generation are typically conditioned on simple class labels (Dhariwal & Nichol, 2021) or textual inputs (Saharia et al., 2022). Pre-trained diffusion models were employed for inverse image problems (Kawar et al., 2022), and different conditioning approaches were compared for score-based models on similar tasks (Batzolis et al., 2021). An unconditional diffusion model has also been combined with inverse problem solving for medical applications (Song et al., 2022). For an in-depth review of diffusion approaches we refer to Yang et al. (2024).

Diffusion for Fluids and Temporal Prediction Selected works have applied diffusion models to temporal prediction tasks like unconditional or text-based video generation, as well as video prediction (e.g. Harvey et al., 2022; Ho et al., 2022; Höppe et al., 2022). These methods typically directly include time as a third dimension or re-use the batch dimension (Blattmann et al., 2023). As a result, autoregressive rollouts are only used to create longer output sequences compared to the training domain, with the drawback that predictions quickly accumulate errors or lose temporal coherence. Few works exist that apply diffusion methods to transient physical processes. To solve inverse physics problems score matching was utilized (Holzschuh et al., 2023), and physics-informed diffusion models for a frame-by-frame super-resolution task for physics simulations exist (Shu et al., 2023). Early steps towards turbulent flows in 3D were taken, via a purely generative diffusion setup based on boundary geometry information (Lienen et al., 2023). Instead of an autoregressive approach, using physical time as a conditioning for diffusion-based fluid field prediction has been investigated, but the authors report stability issues and unphysical predictions as a result (Yang & Sommer, 2023). A multi-step refinement process similar to diffusion models was proposed to improve PDE predictions (Lippe et al., 2023), which is also analyzed below. While it is possible to achieve stability improvements with this approach, it provided little variance in posterior samples and was highly sensitive to hyperparameters in our experiments. Predictor-interpolator schemes inspired by diffusion models were also introduced (Cachay et al., 2023). This method combines a predictor, that equates the diffusion time step with the physical time step of the simulation, with a probabilistic, Bayesian interpolator model. As a result, compared to the direct application of diffusion models analyzed here, this allows for larger time steps and potential performance improvements, while having drawbacks in terms of posterior coverage and temporal coherence.

3. Background on Diffusion Models

A denoising diffusion probabilistic model (DDPM) is a generative model based on a parameterized Markov chain, and contains a fixed forward and a learned reverse process (Ho et al., 2020) over R steps. For any $r \in 0, 1, \ldots, R$, the forward process

$$q(\boldsymbol{x}_r|\boldsymbol{x}_{r-1}) = \mathcal{N}(\boldsymbol{x}_r; \sqrt{1-\beta_r}\boldsymbol{x}_{r-1}, \beta_r \mathbf{I}) \tag{1}$$

incrementally adds Gaussian noise to the original data $\boldsymbol{x}_0$ according to a variance schedule $\beta_1, \ldots, \beta_R$ resulting in the latent variable $\boldsymbol{x}_R$, that corresponds to pure Gaussian noise. The reverse process

$$p_\theta(\boldsymbol{x}_{r-1}|\boldsymbol{x}_r) = \mathcal{N}(\boldsymbol{x}_{r-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_r, r), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}_r, r)) \tag{2}$$

contains learned transitions, i.e. $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\Sigma}_\theta$ are computed by a neural network parameterized by θ given $\boldsymbol{x}_r$ and r. The network is trained via the variational lower bound (ELBO) using reparameterization. During inference the initial latent variable $\boldsymbol{x}_R \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as well as the intermediate diffusion steps are sampled, leading to a probabilistic generation of $\boldsymbol{x}_0$ with a distribution similar to the training data.

To condition the DDPM on information like the initial state and characteristic dimensionless quantities for flow prediction, we employ a concatenation-based conditioning approach (Batzolis et al., 2021): Each element $\boldsymbol{x}_0 = (\boldsymbol{d}_0, \boldsymbol{c}_0)$ of the diffusion process now consists of a data component $\boldsymbol{d}_0$ that is only available during training and a conditioning component $\boldsymbol{c}_0$ that is always given. Correspondingly, the inference task is the conditional prediction $P(\boldsymbol{d}_0|\boldsymbol{c}_0)$. During training, the basic DDPM algorithm remains unchanged as

$$\boldsymbol{x}_r = (\boldsymbol{c}_r, \boldsymbol{d}_r); \quad \boldsymbol{c}_r \sim q(\cdot\,|\boldsymbol{c}_{r-1}); \; \boldsymbol{d}_r \sim q(\cdot\,|\boldsymbol{d}_{r-1}) \quad (3)$$

is still produced by the incremental addition of noise during the forward process. During inference $\boldsymbol{d}_R \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is sampled and processed in the reverse process, while $\boldsymbol{c}_0$ is known and any $\boldsymbol{c}_r$ thus can be obtained from Eq. (1), i.e.,

$$\boldsymbol{x}_r = (\boldsymbol{c}_r, \boldsymbol{d}_r); \quad \boldsymbol{c}_r \sim q(\cdot\,|\boldsymbol{c}_{r-1}); \; \boldsymbol{d}_r \sim p_\theta(\cdot\,|\boldsymbol{x}_{r+1}). \quad (4)$$

Here, $q(\boldsymbol{c}_r|\boldsymbol{c}_{r-1})$ denotes the forward process for $\boldsymbol{c}$, and $\boldsymbol{d}_r \sim p_\theta(\cdot\,|\boldsymbol{x}_{r+1})$ is realized by discarding the prediction of $\boldsymbol{c}_r$ when evaluating p_θ. A visualization of this conditioning technique is shown in Fig. 1. We found the addition of noise to the conditioning, instead of simply using $\boldsymbol{c}_0$ over the entire diffusion process, to be crucial for the temporal stability of the simulation rollout as detailed in Sec. 6.

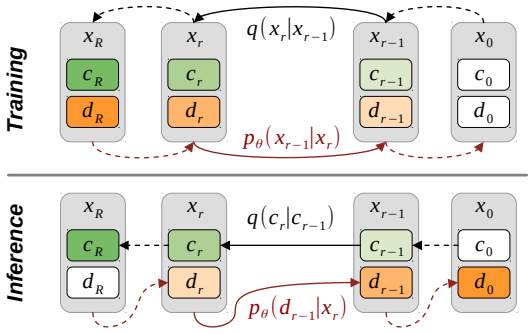

Figure 1. Diffusion conditioning approach with the forward (black) and reverse process (red) during training and inference. White backgrounds for $\boldsymbol{c}$ or $\boldsymbol{d}$ indicate given information, i.e., inputs for each phase. In the context of the autoregressive surrogate simulator, $\boldsymbol{c}_0$ contains information about the simulated process like Reynolds or Mach number, as well as the initial or previous simulation state. $\boldsymbol{d}_0$ contains the next target simulation state during training, and is the resulting prediction of the next state during inference.

4. Flow Prediction Architectures

Our problem setting is the following: a temporal trajectory $s^1, s^2, \ldots, s^T$ of states should be predicted given an initial state s^0. Each s^t consists of dense simulation fields like velocity or pressure, combined with simulation parameters like the Reynolds or Mach number, as constant, spatially expanded channels (see right of Fig. 2). Numerical solvers f iteratively predict $s^t = f(s^{t-1})$, and here we similarly investigate neural surrogate models f_θ with parameters θ to autoregressively predict $s^t = f_\theta(s^{t-1})$, or $s^t \sim f_\theta(s^{t-1})$ for probabilistic methods. In the following, it is important to distinguish this *simulation rollout* via f_θ from the *diffusion rollout* p_θ. The former corresponds to physical time and consists of *simulation-* or *time steps* denoted by $t \in 0, 1, \ldots, T$ superscripts. The latter refers to *diffusion steps* inside the Markov chain, denoted by $r \in 0, 1, \ldots, R$ subscripts as above. To ensure a fair comparison, all models were parameterized with a similar parameter count, and suitable key hyperparameters were determined with a broad search for each architecture. Implementation details for each of the following architectures can be found in App. B.

4.1. Autoregressive Conditional Diffusion Models

To extend the conditional DDPM described in Sec. 3 to temporal tasks, we build on autoregressive single-step prediction, with k previous steps: $s^t \sim f_\theta(\cdot\,|s^{t-k}, \ldots, s^{t-1})$.

Training and Inference f_θ is trained in the following manner: Given a data set with different physical simulation trajectories, a random simulation state $s^t \in s^0, s^1, \ldots, s^T$ is selected from a sequence as the prediction target $\boldsymbol{d}_0$. The corresponding conditioning consists of k previous simulation states $\boldsymbol{c}_0 = (s^{t-k}, \ldots, s^{t-1})$. Next, a random diffusion time step r is sampled, leading to $\boldsymbol{x}_r$ via the forward process. The network learns to predict the added noise level via the ELBO as in the original DDPM (Ho et al., 2020).

The training objective above allows for producing a single subsequent time step as the final output of the diffusion process $\boldsymbol{d}_0$ during inference. We can then employ this single-step prediction for sampling simulation rollouts with arbitrary length by autoregressively reusing generated states as conditioning for the next iteration: For each simulation step, Eq. (4) is unrolled from $\boldsymbol{x}_R^t$ to $\boldsymbol{x}_0^t$ starting from $\boldsymbol{d}_R^t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\boldsymbol{c}_0^t = (s^{t-k}, \ldots, s^{t-1})$. Then, the predicted next time step is $s^t = \boldsymbol{d}_0^t$. This process is visualized in Fig. 2, and we denote such models as *autoregressive conditional diffusion models* (ACDMs) in the following.

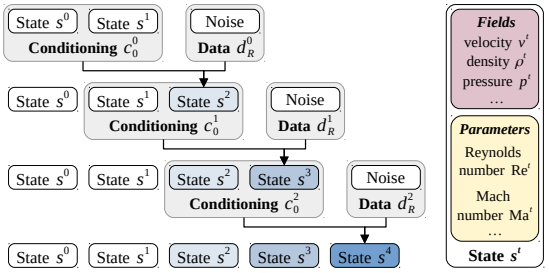

Figure 2. Autoregressive simulation rollout with diffusion models for $k = 2$ input steps (left), and simulation state contents (right).

The motivation for this combination of conditioning and simulation rollout is the hypothesis that perturbations to the

conditioning can be compensated during the diffusion rollout, leading to improved temporal stability. Especially so, when smaller inference errors inevitably accumulate over the course of long simulation rollouts. In addition, the autoregressive rollout ensures that the network produces a temporally coherent trajectory for every step along the inferred sequence. This stands in contrast to explicitly conditioning the DDPM on physical time t, i.e. treating it in the same way as the diffusion step r (see Yang & Sommer, 2023). For a probabilistic model it is especially crucial to have access to previously generated outputs during the prediction, as otherwise temporal coherence can only be achieved via tricks such as fixed spatio-temporal noise patterns.

Implementation We employ a widely used U-Net (based on Ronneberger et al., 2015) with various established architecture modernizations (Dhariwal & Nichol, 2021; Ho et al., 2020), to learn the reverse process. A standard, linear DDPM variance schedule is used. We use $k = 2$ previous steps for the model input, and achieved high-quality samples with 20–100 diffusion steps R, depending on how strongly each setting is conditioned. Combining $\boldsymbol{d}_0$ and $\boldsymbol{c}_0$ to form $\boldsymbol{x}_0$, as well as aggregating multiple states for $\boldsymbol{c}_0$ is achieved via concatenation along the channel dimension.

4.2. U-Net Variants

A crucial question is how much difference the diffusion training itself makes in comparison to more traditional training approaches of the same backbone architecture. Hence, we investigate a range of model variants with identical network architecture to the *ACDM* model, that are optimized with different supervised setups.

Classic Next-step Predictor As a first, simple baseline, the backbone architecture is trained with an MSE loss to directly predict one future simulation state with a single model pass. It is denoted by *U-Net* in the following.

Unrolled Training Several works have reported improvements from unrolling predictions during training (Geneva & Zabaras, 2020; Lusch et al., 2018). We also analyze U-Net architectures with such an unrolled training (*U-Net*$_{ut}$) over m steps, where the gradient is fully backpropagated through the entire training trajectory. This additional complexity during training results in substantial stability improvements, due to a reduced data shift during inference. For this approach, we found $m = 8$ to be ideal across experiments.

Training Noise The usage of training noise was proposed to reduce error accumulation during inference by explicitly learning to compensate errors in the training data (Sanchez-Gonzalez et al., 2020). We investigate adding normally distributed noise to the U-Net input with varying standard deviation n, and denote this model with training noise by *U-Net*$_{tn}$. We found values around $n = 10^{-2}$ to work well.

PDE-Refiner PDE-Refiner is a multi-step refinement process to improve the stability of learned PDE predictions (Lippe et al., 2023). It relies on starting from the predictions of a trained one-step model, and iteratively refining them by adding noise of decreasing variance and denoising the result with the same model. The resulting model is then autoregressively unrolled to form a prediction trajectory. We re-implement this method, closely following the authors' pseudocode, only changing the backbone network to our *U-Net* implementation for a fair comparison against the remaining architectures. PDE-Refiner, denoted by *Refiner* below, relies on two key hyperparameters, the number of refinement steps R, and the minimum noise variance σ. We found this approach to be highly sensitive to both parameters as detailed in App. I.6. Here, we report representative results using $R = 4$ and $\sigma = 10^{-6}$ ($\sigma = 10^{-5}$ for `Iso`), in line with the authors' recommendations (Lippe et al., 2023).

4.3. ResNets and Fourier Neural Operators

As additional popular approaches from the class of direct next-step predictor models, we investigate dilated ResNets (based on Stachenfeld et al., 2022) and Fourier Neural Operators (FNOs) (Li et al., 2021). For the former, the proposed dilated ResNet (*ResNet*$_{dil.}$) as well as the same architecture without dilations (*ResNet*) are included. For the latter, we investigate models using $(16, 8)$ Fourier modes in x- and y-direction (*FNO*$_{16}$), as well as $(32, 16)$ modes (*FNO*$_{32}$).

4.4. Latent-space Transformers

The success of transformer architectures (Vaswani et al., 2017) and their recent application to physics predictions (Geneva & Zabaras, 2022; Han et al., 2021) raises the question how the other approaches fare in comparison to state-of-the-art transformer architectures. Being tailored to sequential processing with a long-term observation horizon, these models operate on a latent space with a reduced size. In contrast, the other investigated architectures by construction operate on the full spatial resolution. Specifically, we test the encoder-processor-decoder architecture from Han et al. (2021) adopted to regular grids via a CNN-encoding, denoted by *TF*$_{MGN}$ below. Furthermore, we provide an improved variant (*TF*$_{Enc}$) that allows to simulate flows with varying parameters over the simulation rollout, and varies key transformer parameters. Compared to *TF*$_{MGN}$ it relies on transformer encoder layers and uses full latent predictions instead of residual predictions. Lastly, we test the transformer-based prediction in conjunction with a probabilistic variational autoencoder, denoted by *TF*$_{VAE}$. All transformer architectures have access to a large number of previous steps and use a rollout schedule in line with the work from Han et al. (2021) during training, however teacher forcing is removed. By default we use a 30 step input window and a training rollout length of 60.

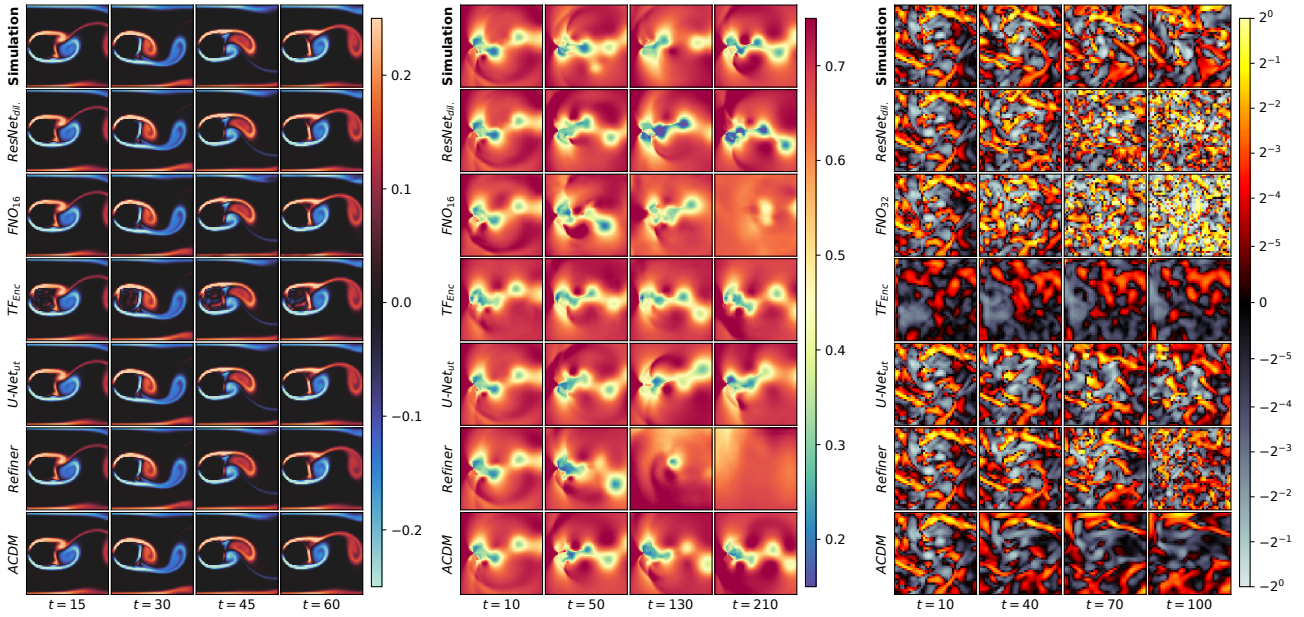

Figure 3. Trajectories from $\texttt{Inc}_{\texttt{high}}$ with $Re = 1000$ (left, vorticity), $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$ (middle, pressure), and $\texttt{Iso}$ with $z = 280$ (right, vorticity). Shown are the simulation reference and key architectures of each model class (also see accompanying videos).

5. Experiments

We quantitatively and qualitatively benchmark the investigated architectures on three flow prediction scenarios with increasing difficulty: (***i***) an incompressible wake flow, (***ii***) a transonic cylinder flow with shock waves, and (***iii***) an isotropic turbulence flow. Test cases for each scenario contain out-of-distribution data via simulation parameters outside of the training data range. Further experimental details are provided in App. A. Examples of the solver trajectory as well as predictions from key architectures from each model class are shown in Fig. 3, with further visualization in App. J and the supplementary videos.

Incompressible Wake Flow Our first case targets incompressible wake flows. These flows already encompass the full complexity of the Navier-Stokes equations with boundary interactions, but due to their direct unsteady periodic nature represent the simplest of our three scenarios. We simulate a fully developed incompressible Karman vortex street behind a cylindrical obstacle with PhiFlow (Holl et al., 2020) for a varying Reynolds number $Re \leq 1000$. The corresponding flows capture the transition from laminar to the onset of turbulence. Models are trained on data from simulation sequences with $Re \in [200, 900]$. We evaluate generalization on the extrapolation test sets $\texttt{Inc}_{\texttt{low}}$ with $Re \in [100, 180]$ for $T = 60$, and $\texttt{Inc}_{\texttt{high}}$ with $Re \in [920, 1000]$ for $T = 60$. While all training is done with constant Re, we add a case with varying Re as a particularly challenging test set: $\texttt{Inc}_{\texttt{var}}$ features a sequence of $T = 250$ steps with a smoothly varying Re from 200 to 900 over the course of the simulation time.

Transonic Cylinder Flow As a significantly more complex scenario we target transonic flows. They require the simulation of a varying density, and exhibit the formation of shock waves interacting with the flow, especially at higher Mach numbers Ma. These properties make the problem highly chaotic and longer prediction rollouts especially challenging. We simulate a fully developed compressible Karman vortex street using SU2 (Economon et al., 2015) with $Re = 10000$, while varying Ma in a transonic regime where shock waves start to occur. Models are trained on sequences with $Ma \in [0.53, 0.63] \cup [0.69, 0.90]$. We evaluate extrapolation on $\texttt{Tra}_{\texttt{ext}}$ with $Ma \in [0.50, 0.52]$ for $T = 60$, interpolation via $\texttt{Tra}_{\texttt{int}}$ with $Ma \in [0.66, 0.68]$ for $T = 60$, and longer rollouts of about 8 vortex shedding periods using $\texttt{Tra}_{\texttt{long}}$ with $Ma \in [0.64, 0.65]$ for $T = 240$.

Isotropic Turbulence As a third scenario we evaluate the inference of planes from 3D isotropic turbulence. This case is inherently difficult, due to its severely underdetermined nature, as the information provided in a 2D plane allows for a large space of possible solutions, depending on the 3D motion outside of the plane. Thus, it is expected that deviations from the reference trajectories occur across methods, and we use $R = 100$ diffusion steps in *ACDM* as a consequence. For this setup, we observed a tradeoff between accuracy and sampling speed, where additional diffusion steps continued to improve prediction quality. As training data, we utilize 2D z-slices of 3D data from the Johns Hopkins Turbulence Database (Perlman et al., 2007). Models are trained on sequences with $z \in [1, 199] \cup [351, 1000]$, and we test on $\texttt{Iso}$ with sequences from $z \in [200, 350]$ for $T = 100$.

Table 1. Quantitative comparison for different network architectures (best and second best results are highlighted for each test set).

	$\texttt{Inc}_{\texttt{low}}$		$\texttt{Inc}_{\texttt{high}}$		$\texttt{Tra}_{\texttt{ext}}$		$\texttt{Tra}_{\texttt{int}}$		Iso	
Method	MSE (10^{-4})	LSiM (10^{-2})	MSE (10^{-5})	LSiM (10^{-2})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-2})	LSiM (10^{-1})
ResNet	10 ± 9.1	17 ± 7.8	16 ± 3.0	5.9 ± 1.6	2.3 ± 0.9	1.4 ± 0.2	1.8 ± 1.0	1.0 ± 0.3	6.7 ± 2.4	9.1 ± 2.2
ResNet$_{dil.}$	1.6 ± 1.8	7.7 ± 5.5	1.5 ± 0.8	2.6 ± 0.7	1.7 ± 1.0	1.2 ± 0.3	1.7 ± 1.4	1.0 ± 0.5	5.7 ± 2.1	8.2 ± 2.0
FNO$_{16}$	2.8 ± 3.1	8.8 ± 7.1	8.9 ± 3.8	2.5 ± 1.2	4.8 ± 1.2	3.4 ± 1.1	5.5 ± 2.6	2.6 ± 1.1	$2m \pm 6m$	15 ± 1.5
FNO$_{32}$	160 ± 50	80 ± 5.4	$1k \pm 140$	57 ± 4.9	4.9 ± 1.9	3.6 ± 0.9	6.8 ± 3.4	3.1 ± 1.1	14 ± 5.3	8.9 ± 1.2
TF$_{MGN}$	5.7 ± 4.3	13 ± 6.4	10 ± 2.9	3.5 ± 0.4	3.9 ± 1.0	1.8 ± 0.3	6.3 ± 4.4	2.2 ± 0.7	8.7 ± 3.8	7.0 ± 2.2
TF$_{Enc}$	1.5 ± 1.7	6.3 ± 4.2	0.6 ± 0.3	1.0 ± 0.3	3.3 ± 1.2	1.8 ± 0.3	6.2 ± 4.2	2.2 ± 0.7	11 ± 5.2	7.2 ± 2.1
TF$_{VAE}$	5.4 ± 5.5	13 ± 7.2	14 ± 19	4.1 ± 1.4	4.1 ± 0.9	2.4 ± 0.2	7.2 ± 3.0	2.7 ± 0.6	11 ± 5.1	7.5 ± 2.1
U-Net	1.0 ± 1.1	5.8 ± 3.2	2.7 ± 0.6	2.6 ± 0.6	3.1 ± 2.1	3.9 ± 2.8	2.3 ± 2.0	3.3 ± 2.8	26 ± 35	11 ± 3.9
U-Net$_{ut}$	0.8 ± 1.1	4.5 ± 4.0	0.2 ± 0.1	0.5 ± 0.2	1.6 ± 0.7	1.1 ± 0.2	1.5 ± 1.5	1.0 ± 0.5	4.5 ± 2.8	2.4 ± 0.5
U-Net$_{tn}$	1.0 ± 1.0	5.6 ± 3.1	0.9 ± 0.6	1.5 ± 0.6	1.4 ± 0.8	1.1 ± 0.3	1.8 ± 1.1	1.0 ± 0.4	3.1 ± 0.9	4.5 ± 2.5
Refiner	1.3 ± 1.4	7.1 ± 4.2	3.5 ± 2.2	2.5 ± 1.0	5.4 ± 2.1	2.3 ± 0.5	7.1 ± 2.1	3.0 ± 1.7	121 ± 200	10.2 ± 3.5
ACDM$_{ncn}$	0.9 ± 0.8	6.6 ± 2.7	5.6 ± 2.6	3.6 ± 1.2	4.1 ± 1.9	1.9 ± 0.6	2.8 ± 1.3	1.7 ± 0.4	18.3 ± 2.5	8.9 ± 1.5
ACDM	1.7 ± 2.2	6.9 ± 5.7	0.8 ± 0.5	1.0 ± 0.3	2.3 ± 1.4	1.3 ± 0.3	2.7 ± 2.1	1.3 ± 0.6	3.7 ± 0.8	3.3 ± 0.7

6. Results

In the following, we benchmark the investigated methods in terms of accuracy, and temporal stability using a range of different metrics. Posterior sampling and the statistical match to the underlying physics are analyzed in Appendices E and F, and Appendices G to I contain additional results and evaluations for various aspects discussed in the following. Unless denoted otherwise, mean and standard deviation over all sequences from each data set, multiple training runs, and multiple random model evaluations are reported. We evaluate two training runs with different random seeds for Iso, and three for Inc and Tra. For the probabilistic methods *TF$_{VAE}$*, *Refiner*, and *ACDM*, five random model evaluations are taken into account per trained model.

Accuracy As the basis for assessing the quality of flow predictions, we first measure direct errors towards the ground truth sequence. We use a mean-squared-error (MSE) and LSiM, a similarity metric for numerical simulations (Kohl et al., 2020). Both metrics are rollout errors, i.e., computed per time step and field, and averaged over the temporal rollout, where lower values indicate better reconstruction accuracy. Table 1 shows the accuracy results, consisting of mean and standard deviation over all sequences from each data set, multiple training runs, and multiple random model evaluations. Errors of models diverging during inference are displayed with factors of 10^3 (k) or 10^6 (m).

For the easiest test case Inc, all model classes can do well, as shown by the overall low errors. The performance of *ResNet$_{dil.}$*, *TF$_{Enc}$*, *U-Net$_{ut}$*, and *ACDM* is quite similar. *FNO$_{16}$* and *Refiner* also work well on $\texttt{Inc}_{\texttt{low}}$, but are slightly less accurate on $\texttt{Inc}_{\texttt{high}}$. On the more complex Tra case, all transformer-based and *FNO* architectures, as well as *Refiner* are already noticeably less accurate, compared to the remaining methods. *ACDM* is on par in terms of error with the established *ResNet* variants, as well as *U-Net* models with unrolling or training noise. However, for the longer rollouts in $\texttt{Tra}_{\texttt{long}}$ the behavior of the architectures is different as temporal stability issues can occur, as discussed below.

On Iso, all models are struggling due to the highly underdetermined nature of this experiment, indicated by the higher errors overall. The transformer-based methods lack accuracy, as the compressed latent representations are unable to capture the high frequency details of this data set. The other architectures with lower relative accuracy accumulate errors over the rollout. This is especially noticeable for *U-Net*, *Refiner*, and the *ResNet* and *FNO* variants. The architectures that remain most stable and accurate are *U-Net$_{ut}$* and *U-Net$_{tn}$*, which are equipped with state of the art stabilization techniques like unrolled training or training noise. Surprisingly, *ACDM* remains fully stable on Iso without modification, and achieves comparable accuracy to *U-Net$_{ut}$* and *U-Net$_{tn}$*. Below, we will mostly focus on the more successful architecture variant in each model class if the behavior is relatively similar, i.e. *ResNet$_{dil.}$* as the superior ResNet model, *TF$_{Enc}$* for the latent-space transformer architectures, and *FNO$_{16}$* (*FNO$_{32}$* for Iso).

The improved performance of *ResNet$_{dil.}$* compared to *ResNet*, and the generally weak results of *FNO* on our more complex tasks confirm the findings from previous work (Stachenfeld et al., 2022). The benefits of unrolling *U-Net* or adding training noise, also become evident on our more complex cases Tra and Iso: while both require additional parameters and training effort, substantial gains in accuracy can be achieved. The regular *U-Net*, despite its network structure being identical to *ACDM*, frequently performs worse than *ACDM* on Tra and Iso. Thus, we in-

clude an ablation on *ACDM*, that behaves similarly to *U-Net* in terms of error propagation: For the $ACDM_{ncn}$ model no conditioning noise is applied, i.e., c_0 is used over the entire diffusion process. This variant performs substantially worse than *ACDM* across cases, as it does not prevent the buildup of errors similar to the *U-Net* or *ResNet* models, due to the tight coupling between conditioning and prediction. This highlights the benefits of creating the next step prediction from scratch, leading to less error propagation and increased temporal stability for *ACDM*.

Temporal Stability A central motivation for analysing diffusion models in the context of transient simulations is the hypothesis that the stochastic training procedure leads to a more robust temporal behavior at inference time. This is especially crucial for practical applications of fluid simulations, where rollouts with thousands of steps are common. To assess temporal stability, we measure the magnitude of the rate of change of s, computed as $\left\|(s^t - s^{t-1})/\Delta t\right\|_1$ for every normalized time step. Compared to a correlation analysis, this metric stays meaningful even for long rollout times on complex data, where sequences can diverge from the specific solution trajectory while still remaining physical. It indicates whether a simulator preserves the expected evolution of states as given by the reference simulation: If predictions explode, the rate of change substantially grows beyond the reference, and if they collapse into an incorrect steady state, the change approaches zero.

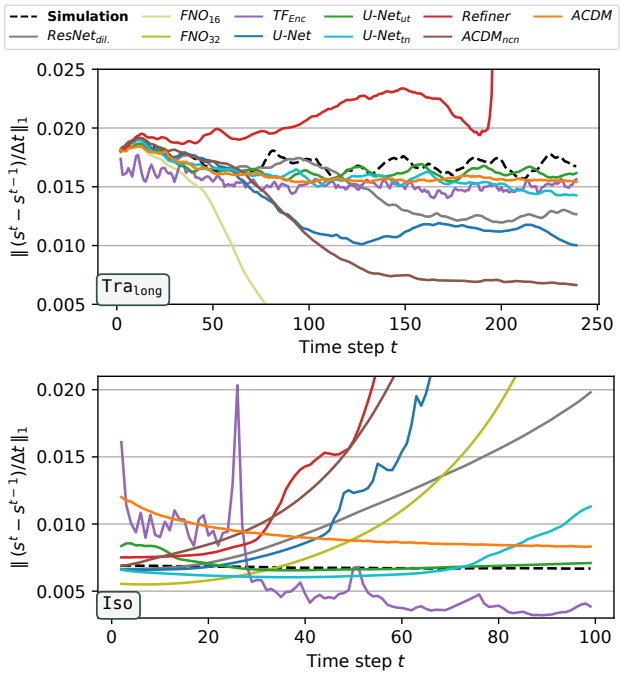

Figure 4. Stability analysis on $\texttt{Tra}_{\texttt{long}}$ (top) and `Iso` (bottom). Standard deviations are omitted for visual clarity.

Figure 4 shows this evaluation for $\texttt{Tra}_{\texttt{long}}$ and `Iso`. Model variants from each class that are omitted perform similar to the included variant in both cases. For $\texttt{Tra}_{\texttt{long}}$, the reference simulation features steady oscillations as given by the main vortex shedding frequency. Note that these vortex shedding oscillations are averaged out over posterior samples and training runs in this evaluation, once architecture diverge from the exact reference trajectory. *Refiner* is highly unstable in this evaluations across its hyperparameters. Even though some trained models and samples are fully stable, the temporal stability on average is worst across the investigated architectures. Next-step predictor architectures like $ResNet_{dil.}$, FNO_{16}, *U-Net* diverge at different points during the $\texttt{Tra}_{\texttt{long}}$ rollout and mostly settle into a stable but wrong state of a mean flow prediction without vortices. $ACDM_{ncn}$ which does not explicitly target error mitigation performs similar as well. TF_{Enc} generally remains stable due to the long training rollout, indicated by a mostly constant rate of change. However, it exhibits minor temporal inconsistencies and slightly undershoots compared to the reference, most likely due to temporal updates being performed suboptimally in the latent space. $U\text{-}Net_{tn}$ is stable, however early deterioration signs are visible towards the end of the trajectory. *ACDM* and $U\text{-}Net_{ut}$ remain fully stable, even for many additional rollouts steps.

At the bottom in Fig. 4, the stability is evaluated for the `Iso` experiment. Its isotropic nature in combination with forcing leads to an almost constant rate of change over time for the reference simulation. All methods struggle to replicate this accurately due the highly underdetermined learning task. In addition to issues with the reconstruction quality, TF_{Enc} exhibits undesirable spikes corresponding to their temporal prediction window of $k = 25$ previous steps. It also undershoots after one rollout window, making it highly undesirable for this task. Similar as observed on $\texttt{Tra}_{\texttt{long}}$, next-step predictor architectures like $ResNet_{dil.}$, FNO_{32}, and *U-Net* initially predict a quite accurate rate of change, but diverge at different points over the rollout. *Refiner* is also highly unstable across trained models and samples. Here, the common failure mode across models is an incorrect addition of energy to the system that causes a quick and significant divergence from the reference.

$U\text{-}Net_{ut}$ and $U\text{-}Net_{tn}$ show the best temporal stability in this experiment, however both exhibit minor issues at the start or end of the simulation rollout. For $U\text{-}Net_{ut}$, this can be further mitigated via a learning schedule to incrementally increase the rollout during training, or by pre-training with few rollout steps. Despite the initially slightly larger rate of change, and the decay corresponding to an overly dissipative prediction, *ACDM* fares well and remains stable over the simulation rollout. For this case, we did observe mild temporal coherence issues in the vorticity computed from the *ACDM* velocity predictions, which are analyzed

Table 2. Overview with our assessment of advantages and drawbacks for different flow prediction approaches at similar training memory (✓✓✓: excellent, ✓✓: good, ✓: acceptable, ✗: suboptimal, ✗✗: bad, N/A: not available, —: not investigated in detail)

Aspect	Latent-space Transformer	FNO	Dilated ResNet	U-Net	U-Net (train. noise)	U-Net (unrolled)	PDE Refiner	Autoreg. Diffusion
Training Speed	✓✓	✓✓✓	✗✗	✓	✓	✗✗	✓	✓
Inference Speed	✓✓✓	✓✓	✓✓	✓	✓	✓	✗	✗✗
Accuracy (`Inc`)	✓✓✓	✓	✓✓✓	✓✓	✓✓✓	✓✓✓	✓✓	✓✓✓
Accuracy (`Tra`)	✓	✓	✓✓✓	✓	✓✓✓	✓✓✓	✓	✓✓✓
Accuracy (`Iso`)	✗	✗✗	✓	✗✗	✓✓✓	✓✓✓	✗✗	✓✓✓
Posterior Sample Diversity	✗ (VAE)	N/A	N/A	N/A	N/A	N/A	✓	✓✓✓
Temporal Stability (`Inc`)	✓✓✓	✓✓✓	✓✓	✓✓	✓✓✓	✓✓✓	✓✓✓	✓✓✓
Temporal Stability (`Tra`)	✓✓	✗	✓	✗	✓✓	✓✓✓	✓	✓✓✓
Temporal Stability (`Iso`)	✗✗	✗	✗	✗✗	✓✓	✓✓✓	✗	✓✓
Key Hyperparameters	Latent Size	Fourier Modes	Dilation Rates	# Up/Down Blocks	Noise Variance	Rollout Length	Diff. Steps, Noise Var.	Diffusion Steps
Key Hyperparameter Stability	—	✗	—	—	✗	✓	✗✗	✓✓

in more detail in App. H. Nevertheless, these experiments clearly show the increased error tolerance of *ACDM* compared to $ACDM_{ncn}$, as the latter performs very similar to *U-Net* for both experiments in Fig. 4, while *ACDM* remains fully stable.

In an additional stability evaluation (see App. G), we unrolled the three most stable architectures $U\text{-}Net_{ut}$, $U\text{-}Net_{tn}$, and *ACDM* for $T = 200\,000$ steps to investigate extremely long prediction horizons which are highly desirable for e.g. climate prediction models (Watt-Meyer et al., 2023). All architectures remained fully stable and statistically accurate on extended sequences from $\texttt{Inc}_{\texttt{high}}$. In addition, we considered more challenging extended sequences from $\texttt{Tra}_{\texttt{ext}}$: while one training run from $U\text{-}Net_{tn}$ did diverge, all runs from *ACDM* and $U\text{-}Net_{ut}$ were fully stable across the entire prediction horizon, indicating their potential for future applications.

Discussion and Overview We summarize the findings of the performed experiments in Tab. 2, and include additional results for performance and posterior sampling from Appendices C and E. In general, latent-space transformers and *FNO* variants are fast to train and evaluate. As they use an identical backbone model, *U-Net*, $U\text{-}Net_{ut}$, and $U\text{-}Net_{tn}$ have the same inference speed. Note however, that $U\text{-}Net_{ut}$ requires substantially more time and memory during training due to the additional rollout steps. Diffusion-based models like PDE-Refiner and *ACDM* have an inference slow-down factor roughly proportional to the number of diffusion steps R, i.e. 2–8 for PDE-Refiner and 20–100 for *ACDM*. Furthermore, the stability with respect to key hyperparameters for some architectures is reported, experiments for which can be found in App. I. To summarize, we draw the following main conclusions:

(***i***) Latent-space transformers are highly accurate and stable, if the input space can be compressed easily.

(***ii***) Next-step predictors like $ResNet_{dil.}$, *FNO*, and *U-Net* work very well for simple tasks, but require explicit stabilization techniques for more complex cases.

(***iii***) If training *and* inference performance are of concern, training noise can provide substantial stabilization.

(***iv***) Unrolling is resource-intensive during training and requires tuning of the rollout length, but pays off in terms of improved accuracy and stability.

(***v***) Autoregressive diffusion models like *ACDM* are as accurate and stable as unrolled training at lower training cost, but are expensive during inference.

(***vi***) PDE-Refiner can improve the temporal stability compared to *U-Net*, however for complex cases it is very sensitive to repeated sampling or training, and its hyperparameters.

(***vii***) If accurate and diverse posterior samples are required, *ACDM* models are the ideal option.

7. Conclusion and Future Work

We investigated the attractiveness of autoregressive conditional diffusion models for the simulation of complex flow phenomena. Our results show that using even a simple diffusion-based approach is on par with established stabilization approaches, while at the same time enabling probabilistic inference. In the future, the inference performance of ACDMs can be improved, e.g., via better sampling procedures such as distillation (Salimans & Ho, 2022). Naturally, considering other PDEs, or larger, three-dimensional flows is likewise a highly interesting direction. For the latter, single-step diffusion approaches are particularly attractive, as they avoid the substantial costs of temporal training rollouts (Sirignano et al., 2020) to achieve stability.

Acknowledgments

This work was supported by the ERC Consolidator Grant *SpaTe* (CoG-2019-863850). We would also like to thank Björn List and Benjamin Holzschuh for helpful discussions and comments during the creation of this work.

References

de Avila Belbute-Peres, F., Economon, T. D., and Kolter, J. Z. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119, pp. 2402–2411, 2020. URL http://proceedings.mlr.press/v119/de-avila-belbute-peres20a.html.

Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. Learning data driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019. ISSN 0027-8424, 1091-6490. doi:10.1073/pnas.1814058116.

Batzolis, G., Stanczuk, J., Schönlieb, C.-B., and Etmann, C. Conditional image generation with score-based diffusion models. *arXiv*, 2021. doi:10.48550/arXiv.2111.13606.

Blattmann, A., Rombach, R., Ling, H., Dockhorn, T., Kim, S. W., Fidler, S., and Kreis, K. Align your latents: High-resolution video synthesis with latent diffusion models. In *2023 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22563–22575, 2023. doi:10.1109/CVPR52729.2023.02161.

Brandstetter, J., Worrall, D. E., and Welling, M. Message passing neural pde solvers. In *10th International Conference on Learning Representations (ICLR 2022)*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Cachay, S. R., Zhao, B., Joren, H., and Yu, R. Dyffusion: A dynamics-informed diffusion model for spatiotemporal forecasting. In *Advances in Neural Information Processing Systems 36*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/8df90a1440ce782d1f5607b7a38f2531-Abstract-Conference.html.

Deo, I. K., Gao, R., and Jaiman, R. Combined space–time reduced-order model with three-dimensional deep convolution for extrapolating fluid dynamics. *Physics of Fluids*, 35(4):043606, 2023. ISSN 1070-6631. doi:10.1063/5.0145071.

Dhariwal, P. and Nichol, A. Q. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems 34*, pp. 8780–8794, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/49ad23d1ec9fa4bd8d77d02681df5cfa-Abstract.html.

Dosovitskiy, A. and Brox, T. Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems 29*, pp. 658–666, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/371bce7dc83817b7893bcdeed13799b5-Abstract.html.

Dryden, H. L. A review of the statistical theory of turbulence. *Quarterly of Applied Mathematics*, 1(1):7–42, 1943. ISSN 0033569X, 15524485. URL http://www.jstor.org/stable/43633324.

Economon, T., Palacios, F., Copeland, S., Lukaczyk, T., and Alonso, J. Su2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54:1–19, 2015. doi:10.2514/1.J053813.

Geneva, N. and Zabaras, N. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403, 2020. doi:10.1016/j.jcp.2019.109056.

Geneva, N. and Zabaras, N. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022. doi:10.1016/j.neunet.2021.11.022.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

Han, X., Gao, H., Pfaff, T., Wang, J.-X., and Liu, L. Predicting physics in mesh-reduced space with temporal attention. In *10th International Conference on Learning Representations (ICLR 2022)*, 2021. URL https://openreview.net/forum?id=XctLdNfCmP.

Harvey, W., Naderiparizi, S., Masrani, V., Weilbach, C., and Wood, F. Flexible diffusion modeling of long videos. In *Advances in Neural Information Processing Systems 35*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b2fe1ee8d936ac08dd26f2ff58986c8f-Abstract-Conference.html.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi:10.1109/CVPR.2016.90.

Hemmasian, A. P. and Farimani, A. B. Reduced-order modeling of fluid flows with transformers. *Physics of Fluids*, 35(5):057126, 2023. ISSN 1070-6631. doi:10.1063/5.0151515.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html.

Ho, J., Salimans, T., Gritsenko, A. A., Chan, W., Norouzi, M., and Fleet, D. J. Video diffusion models. In *Advances in Neural Information Processing Systems 35*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/39235c56aef13fb05a6adc95eb9d8d66-Abstract-Conference.html.

Holl, P., Thuerey, N., and Koltun, V. Learning to control pdes with differentiable physics. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020. URL https://openreview.net/forum?id=HyeSin4FPB.

Holzschuh, B. J., Vegetti, S., and Thuerey, N. Solving inverse physics problems with score matching. In *Advances in Neural Information Processing Systems 36*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/c2f2230abc7ccf669f403be881d3ffb7-Abstract-Conference.html.

Höppe, T., Mehrjou, A., Bauer, S., Nielsen, D., and Dittadi, A. Diffusion models for video prediction and infilling. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=1f0lr4AYM6.

Hu, T. and Liao, S. On the risks of using double precision in numerical simulations of spatio-temporal chaos. *Journal of Computational Physics*, 418:109629, 2020. doi:10.1016/j.jcp.2020.109629.

Hyvärinen, A. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005. URL http://jmlr.org/papers/v6/hyvarinen05a.html.

Johnson, J., Alahi, A., and Fei-Fei, L. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision - ECCV 2016*, volume 9906, pp. 694–711, 2016. doi:10.1007/978-3-319-46475-6_43.

Kawar, B., Elad, M., Ermon, S., and Song, J. Denoising diffusion restoration models. In *Advances in Neural Information Processing Systems 35*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/95504595b6169131b6ed6cd72eb05616-Abstract-Conference.html.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M. H., and Solenthaler, B. Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38 (2):59–70, 2019. doi:10.1111/cgf.13619.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR 2015)*, 2015. URL http://arxiv.org/abs/1412.6980.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations (ICLR 2014)*, 2014. URL http://arxiv.org/abs/1312.6114.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021. doi:10.1073/pnas.2101784118.

Kohl, G., Um, K., and Thuerey, N. Learning similarity metrics for numerical simulations. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119, pp. 5349–5360, 2020. URL http://proceedings.mlr.press/v119/kohl20a.html.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. In *9th International Conference on Learning Representations (ICLR 2021)*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Lienen, M., Hansen-Palmus, J., Lüdke, D., and Günnemann, S. From zero to turbulence: Generative modeling for 3d flow simulation. *arXiv*, 2023. doi:10.48550/arXiv.2306.01776.

Lippe, P., Veeling, B., Perdikaris, P., Turner, R. E., and Brandstetter, J. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. In *Advances in Neural Information Processing Systems 36*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d529b943af3dba734f8a7d49efcb6d09-Abstract-Conference.html.

List, B., Chen, L.-W., and Thuerey, N. Learned turbulence modelling with differentiable fluid solvers: Physics-based loss functions and optimisation horizons. *Journal of Fluid Mechanics*, 949:A25, 2022. doi:10.1017/jfm.2022.738.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. In *2022 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11966–11976, 2022. doi:10.1109/CVPR52688.2022.01167.

Lusch, B., Kutz, J. N., and Brunton, S. L. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018. ISSN 2041-1723. doi:10.1038/s41467-018-07210-0.

Moin, P. and Mahesh, K. Direct numerical simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics*, 30(1): 539–578, 1998. doi:10.1146/annurev.fluid.30.1.539.

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139, pp. 8162–8171, 2021. URL http://proceedings.mlr.press/v139/nichol21a.html.

Olufsen, M. S., Peskin, C. S., Kim, W. Y., Pedersen, E. M., Nadim, A., and Larsen, J. Numerical simulation and experimental validation of blood flow in arteries with structured-tree outflow conditions. *Annals of Biomedical Engineering*, 28(11):1281–1299, 2000. doi:10.1114/1.1326031.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, 2019. doi:10.48550/arXiv.1912.01703.

Perlman, E., Burns, R., Li, Y., and Meneveau, C. Data exploration of turbulence simulations using a database cluster. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing*, pp. 1–11, 2007. doi:10.1145/1362622.1362654.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. In *9th International Conference on Learning Representations (ICLR 2021)*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Pope, S. *Turbulent Flows*. Cambridge University Press, 2000. ISBN 978-0-511-84053-1. doi:10.1017/CBO9780511840531.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351, pp. 234–241, 2015. doi:10.1007/978-3-319-24574-4_28.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, S. K. S., Lopes, R. G., Ayan, B. K., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. Photorealistic text-to-image diffusion models with deep language understanding. In *Advances in Neural Information Processing Systems 35*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/ec795aeadae0b7d230fa35cbaf04c041-Abstract-Conference.html.

Salimans, T. and Ho, J. Progressive distillation for fast sampling of diffusion models. In *10th International Conference on Learning Representations (ICLR 2022)*, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119, pp. 8459–8468, 2020. URL http://proceedings.mlr.press/v119/sanchez-gonzalez20a.html.

Shen, Z., Zhang, M., Zhao, H., Yi, S., and Li, H. Efficient attention: Attention with linear complexities. In *IEEE Winter Conference on Applications of Computer Vision*, pp. 3530–3538, 2021. doi:10.1109/WACV48630.2021.00357.

Shu, D., Li, Z., and Farimani, A. B. A physics-informed diffusion model for high-fidelity flow field reconstruction. *Journal of Computational Physics*, 478:111972, 2023. doi:10.1016/j.jcp.2023.111972.

Sirignano, J. A., MacArt, J. F., and Freund, J. B. Dpm: A deep learning pde augmentation method with application to large-eddy simulation. *Journal of Computational Physics*, 423:109811, 2020. doi:10.1016/j.jcp.2020.109811.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, volume 37, pp. 2256–2265, 2015. URL http://proceedings.mlr.press/v37/sohl-dickstein15.html.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. In *9th International Conference on Learning Representations (ICLR 2021)*, 2021a. URL https://openreview.net/forum?id=St1giarCHLP.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *9th International Conference on Learning Representations (ICLR 2021)*, 2021b. URL https://openreview.net/forum?id=PxTIG12RRHS.

Song, Y., Shen, L., Xing, L., and Ermon, S. Solving inverse problems in medical imaging with score-based generative models. In *10th International Conference on Learning Representations (ICLR 2022)*, 2022. URL https://openreview.net/forum?id=vaRCHVj0uGI.

Spalart, P., Deck, S., Shur, M. L., Squires, K., Strelets, M., and Travin, A. A new version of detached-eddy simulation, resistant to ambiguous grid densities. *Theoretical and Computational Fluid Dynamics*, 20(3):181–195, 2006. ISSN 1432-2250. doi:10.1007/s00162-006-0015-0.

Stachenfeld, K. L., Fielding, D. B., Kochkov, D., Cranmer, M. D., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P. W., and Sanchez-Gonzalez, A. Learned coarse models for efficient turbulence simulation. In *10th International Conference on Learning Representations (ICLR 2022)*, 2022. URL https://openreview.net/forum?id=msRBojTz-Nh.

Thuerey, N., Weissenow, K., Prantl, L., and Hu, X. Deep learning methods for reynolds-averaged navier-stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020. doi:10.2514/1.J058291.

Um, K., Brand, R., Fei, Y., Holl, P., and Thuerey, N. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/43e4e6a6f341e00671e123714de019a8-Abstract.html.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Verma, S., Novati, G., and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018. doi:10.1073/pnas.1800923115.

Wang, R., Kashinath, K., Mustafa, M., Albert, A., and Yu, R. Towards physics-informed deep learning for turbulent flow prediction. In *26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1457–1466, 2020. doi:10.1145/3394486.3403198.

Watt-Meyer, O., Dresdner, G., McGibbon, J., Clark, S. K., Henn, B., Duncan, J., Brenowitz, N. D., Kashinath, K., Pritchard, M. S., Bonev, B., Peters, M. E., and Bretherton, C. S. Ace: A fast, skillful learned global atmospheric model for climate prediction. *arXiv*, 2023. doi:10.48550/arXiv.2310.02074.

Wiewel, S., Becher, M., and Thuerey, N. Latent space physics: Towards learning the temporal evolution of fluid flow. *Computer Graphics Forum*, 38(2):71–82, 2019. ISSN 1467-8659. doi:10.1111/cgf.13620.

Wu, Y. and He, K. Group normalization. In *Computer Vision - ECCV 2018*, volume 11217, pp. 3–19, 2018. doi:10.1007/978-3-030-01261-8_1.

Wyngaard, J. C. Atmospheric turbulence. *Annual Review of Fluid Mechanics*, 24(1):205–234, 1992. doi:10.1146/annurev.fl.24.010192.001225.

Yang, G. and Sommer, S. A denoising diffusion model for fluid field prediction. *arXiv*, 2023. doi:10.48550/arXiv.2301.11661.

Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Zhang, W., Cui, B., and Yang, M.-H. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4):105:1–105:39, 2024. doi:10.1145/3626235.

A. Data Details

In the following, we provide details for each simulation setup: the incompressible wake flow `Inc` in App. A.1, the transonic cylinder flow `Tra` in App. A.2, and the isotropic turbulence `Iso` in App. A.3. Further details can be found in our source code at `https://github.com/tum-pbs/autoreg-pde-diffusion`.

A.1. Incompressible Flow Simulation

To create the incompressible cylinder flow we employ the fluid solver PhiFlow[1] (Holl et al., 2020). Velocity data is stored on a staggered grid, we employ an advection scheme based on the MacCormack method, and use the adaptive conjugate gradient method as a pressure solver. We enforce a given Reynolds number in $[100, 1000]$ via an explicit diffusion step.

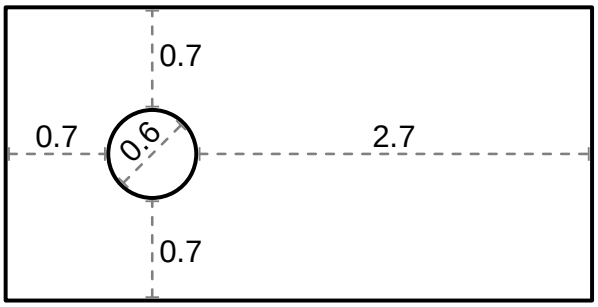

Figure 5. Simulation domain for incompressible flow simulation.

Our domain setup is illustrated in Fig. 5. We use Neumann boundary conditions in vertical x-direction of the domain and around the cylinder, and a Dirichlet boundary condition for the outflow on the right of the domain. For the inflow on the left of the domain we prescribe a fixed freestream velocity of $\binom{0}{0.5}$ during the simulation. To get oscillations started, the y-component of this velocity is replaced with $0.5 \cdot (\cos(\pi \cdot x) + 1)$, where x denotes normalized vertical domain coordinates in $[0, 1]$, during a warmup of 20 time steps. We run and export the simulation for 1300 iterations at time step 0.05, using data after a suitable warmup period $t > 300$. The spatial domain discretization is 256×128, but we train and evaluate models on a reduced resolution via downsampling the data to 128×64. Velocities are resampled to a regular grid before exporting, and pressure values are exported directly. In addition, we normalize all fields and scalar components to a standard normal distribution. The velocity is normalized in terms of magnitude. During inference we do not evaluate the cylinder area; i.e., all values inside the cylinder are set to zero via a multiplicative binary mask before every evaluation or loss computation.

We generated a data set of 91 sequences with Reynolds number $Re \in \{100, 110, \dots, 990, 1000\}$. Running and exporting the simulations on a machine with an NVIDIA GeForce GTX 1080 Ti GPU and an Intel Core i7-6850k CPU with 6 cores at 3.6 GHz took about 5 days. Models are trained using the data of 81 sequences with $Re \in \{200, 210, \dots, 890, 900\}$ for $t \in [800, 1300]$. Training and test sequences employ a temporal stride of 2. As test sets we use:

- $\texttt{Inc}_{\texttt{low}}$: five sequences with $t \in [1000, 1120)$ and thus $T = 60$, for $Re \in \{100, 120, 140, 160, 180\}$.
- $\texttt{Inc}_{\texttt{high}}$: five sequences with $t \in [1000, 1120)$ and thus $T = 60$, for $Re \in \{920, 940, 960, 980, 1000\}$.
- $\texttt{Inc}_{\texttt{var}}$: one sequence for $t \in [300, 800)$ with $T = 250$, and a smoothly varying Re from 200 to 900 during the simulation. This is achieved via linearly interpolating the diffusivity to the corresponding value at each time step.

For the $\texttt{Inc}_{\texttt{var}}$ test set, we replace the model predictions of Re that are learned to be constant for *ACDM*, *U-Net*, *ResNet*, and *FNO* with the linearly varying Reynolds numbers over the simulation rollout during inference. The transformer-based methods TF_{Enc} and TF_{VAE} receive all scalar simulation parameters as an additional input to the latent space for each iteration of the latent processor. Note that the architectural design of TF_{MGN} does not allow for varying simulation parameters over the rollout, as only one fixed parameter embedding is provided as a first input step for the latent processor, i.e. the model is expected to diverge quickly due to the data shift in such cases.

A.2. Transonic Flow Simulation

To create the transonic cylinder flow we use the simulation framework SU2[2] (Economon et al., 2015). We employ the delayed detached eddy simulation model (SA-DDES) for turbulence closure, which is derived from the one-equation Spalart-Allmaras model (Spalart et al., 2006). By modifying the length scale, the model behaves like RANS for the attached flow in the near wall region and resolves the detached flows in the other regions. No-slip and adiabatic conditions are applied on the cylinder surface. The farfield boundary conditions are treated by local, one-dimensional Riemann-invariants. The governing equations are numerically solved by the finite-volume method. Spatial gradients are computed with weighted least squares, and the biconjugate gradient stabilized method (BiCGSTAB) is used as the implicit linear solver. For the freestream velocity we enforce a given Mach number in $[0.5, 0.9]$ while keeping the Reynolds number at a constant value of 10^4.

[1] `https://github.com/tum-pbs/PhiFlow`

[2] `https://su2code.github.io`

To prevent issues with shockwaves from the initial flow phase, we first compute a steady RANS solution for each case for 1000 solver iterations and use that as the initialization for the unsteady simulation. We run the unsteady simulation for 150 000 iterations overall, and use every 50$^{\text{th}}$ step once the vortex street is fully developed after the first 100 000 iterations. This leads to $T = 1000$ exported steps with velocity, density, and pressure fields. The non-dimensional time step for each simulation is $0.002 * \tilde{D}/\tilde{U}_\infty$, where $\tilde{D}$ is the dimensional cylinder diameter, and $\tilde{U}_\infty$ the free-stream velocity magnitude.

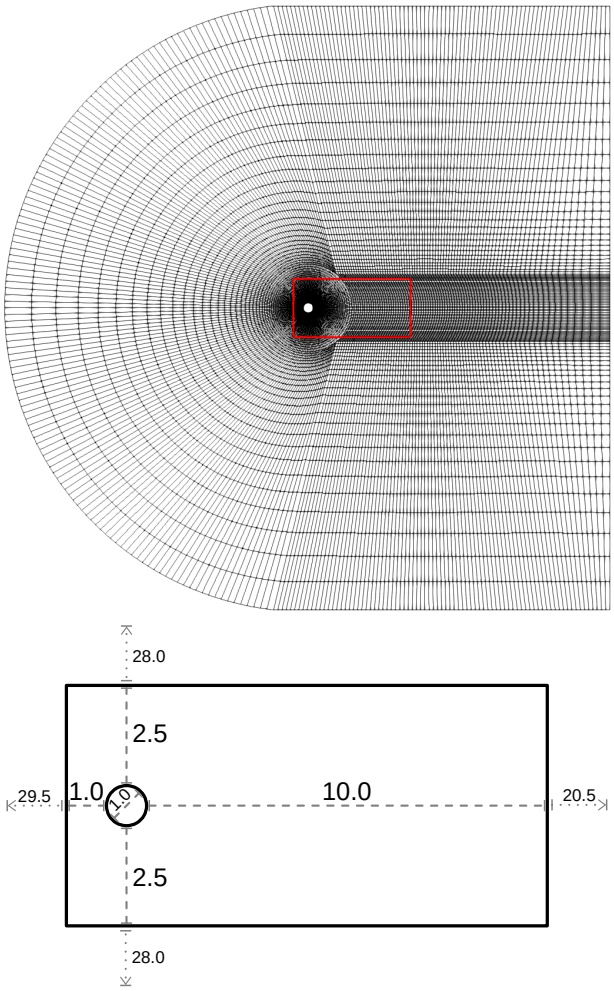

Figure 6. Full simulation mesh with highlighted resampling area (top) and resampling domain setup (bottom) for the transonic flow simulation.

The computational mesh is illustrated on the top in Fig. 6. Inference is focused on the near field region around the obstacle (marked in red on the top, and shown in detail on the bottom). To interpolate from the original mesh to the resampled training and testing domain, which is a regular, Cartesian grid with resolution 128×64, we use an interpolation based on radial basis functions. It employs a linear basis function across the 5 nearest data points of the original mesh. In terms of field normalization and masking the cylinder area during inference, we treat this case in the same way as described in App. A.1.

We created a data set of 41 sequences with Mach number $Ma \in \{0.5, 0.51, \dots, 0.89, 0.90\}$ at Reynolds number 10^4 with the $T = 1000$ exported steps each. We sequentially ran the simulations on one CPU cluster node that contains 28 Intel Xeon E5-2690 v3 CPU cores at 2.6 GHz in about 5 days. Each simulation was computed in parallel with 56 threads, and one separate thread simultaneously resampled and processed the simulation outputs online during the simulation. All models are trained on the data of 33 sequences with $Ma \in \{0.53, 0.54, \dots, 0.62, 0.63\} \cup \{0.69, 0.70, \dots, 0.89, 0.90\}$. Training and test sequences use a temporal stride of 2. The used test cases for this compressible, transonic flow setup are:

- $\texttt{Tra}_{\texttt{ext}}$: six sequences from $Ma \in \{0.50, 0.51, 0.52\}$, for $t \in [500, 620)$ and for $t \in [620, 740)$ with $T = 60$.
- $\texttt{Tra}_{\texttt{int}}$: six sequences from $Ma \in \{0.66, 0.67, 0.68\}$, for $t \in [500, 620)$ and for $t \in [620, 740)$ with $T = 60$.
- $\texttt{Tra}_{\texttt{long}}$: four sequences from $Ma \in \{0.64, 0.65\}$, for $t \in [0, 480)$ and for $t \in [480, 960)$ with $T = 240$.

A.3. Isotropic Turbulence

For the isotropic turbulence experiment, we make use of the 3D *isotropic1024coarse* simulation from the Johns Hopkins Turbulence Database[3] (Perlman et al., 2007). It contains simulations of forced turbulence with a direct numerical simulation (DNS) using a pseudo-spectral method on 1024^3 nodes for 5028 time steps. The database allows for direct download queries of parameterized simulation cutouts; filtering and interpolation are already provided. We utilize sequences of individual 2D slices with a spatio-temporal starting point of $(s_x, s_y, s_z, s_t) = (1, 1, z, 1)$ and end point of $(e_x, e_y, e_z, e_t) = (256, 128, z + 1, 1000)$ for different values of z. A spatial striding of 2 leads to the training and evaluation resolution of 128×64. We use the pressure, as well as the velocity field including the velocity z-component. We normalize all fields to a standard normal distribution before training and inference. In this case, the velocity components are normalized individually, which is statistically comparable to a normalization in terms of magnitude for isotropic turbulence.

We utilize 1000 sequences with $z \in \{1, 2, \dots, 999, 1000\}$ and $T = 1000$. Models are trained on 849 sequences with $z \in \{1, 2, \dots, 198, 199\} \cup \{351, 352, \dots, 999, 1000\}$. The test set in this case is $\texttt{Iso}$ using 16 sequences from $z \in \{200, 210, \dots, 340, 350\}$ for $t \in [500, 600)$, meaning $T = 100$.

[3] https://turbulence.pha.jhu.edu/

B. Implementation and Model Details

Using the data generated with the techniques described above, the deep learning aspects of this work are implemented in PyTorch (Paszke et al., 2019). For every model we optimize network weights using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 10^{-4} (using $\beta_1 = 0.9$ and $\beta_2 = 0.999$), where the batch size is chosen as 64 by default. If models would exceed the available GPU memory, the batch size is reduced accordingly. For each epoch, the long training sequences are split into shorter parts according to the required training sequence length for each model and the temporal strides described in App. A. To prevent issues with a bias towards certain initial states, the start (and corresponding end) of each training sequence is randomly shifted forwards or backwards in time by half the sequence length every time the sequence is loaded. This is especially crucial for the oscillating cylinder flows when training models with longer rollouts. For instance, training a model with a training rollout length of 60 steps on a data set that contains vortex shedding oscillations with a period of 30 steps would lead to a correlation between certain vortex arrangements and the temporal position in the rollout during training (and inference). This could potentially lead to generalization problems when the model is confronted with a different vortex arrangement than expected at a certain time point in the rollout. The sequences for each test set are used directly without further modifications. In the following, we provide architectural and training details for the different model architectures discussed in the main paper.

B.1. ACDM Implementation

For the *ACDM* models, we employ a "modern" U-Net architecture commonly used for diffusion models: The setup at its core follows the traditional U-Net architecture (Ronneberger et al., 2015) with an initial convolution layer, several downsampling blocks, one bottleneck block, and several upsampling blocks followed by a final convolution layer. The downsampling and upsampling block at one resolution are connected via skip connections in addition to the connections through lower layers. The modernizations mainly affect the number and composition of the blocks: We use three feature map resolutions (128×64, 64×32, and 32×16), i.e. three down- and three upsamling blocks, with a constant number of channels of 128 at each resolution level. The down- and upsampling block at each level consists of two ConvNeXt blocks (Liu et al., 2022) and a linear attention layer (Shen et al., 2021). The bottleneck block uses a regular multi-head self-attention layer (Vaswani et al., 2017) instead. As proposed by Ho et al. (2020), we:

- use group normalization (Wu & He, 2018) throughout the blocks,
- use a diffusion time embedding for the diffusion step r via a Transformer sinusoidal position embedding layer (Vaswani et al., 2017) combined with an MLP consisting of two fully connected layers, that is added to the input of every ConvNeXt block,
- train the model via reparameterization,
- and employ a linear variance schedule.

Since the variance hyperparameters provided by Ho et al. (2020) only work for a large number of diffusion steps R, we adjust them accordingly to fewer diffusion steps: $\beta_0 = 10^{-4} * (500/R)$ and $\beta_R = 0.02 * (500/R)$. We generally found $R = 20$ to be sufficient on the strongly conditioned data set `Inc` and `Tra`, but on the highly complex `Iso` data, *ACDM* showed improvements up to about $R = 100$. The same value of R is used during training and inference. In early exploration runs, we found $k = 2$ input steps to show slightly better performance compared to $k = 1$ used by *U-Net* below, and kept this choice for consistency across diffusion evaluations. However, the differences for changing the number of input steps from $k \in \{1, 2, 3, 4\}$ are minor compared to the performance difference between architectures. The resulting models are trained for 3100 epochs on `Inc` and `Tra`, and 100 epochs on `Iso`. All setups use a batch size of 64 during training, and employ a Huber loss, which worked better than an MSE loss. However, the performance difference between the losses are marginal, compared to the difference between architectures.

For the *ACDM*$_{ncn}$ variants, we leave all these architecture and training parameters untouched, and only change the conditioning integration: Instead of adding noise to $\boldsymbol{c}_0$ in the forward and reverse diffusion process at training and inference time, $\boldsymbol{c}_0$ is used without alterations over the entire diffusion rollout.

B.2. Implementation of U-Net and Variants with Stabilization

For the implementation of *U-Net* we use an identical U-Net architecture as described above in App. B.1. The only difference being that the diffusion time embeddings are not necessary. The resulting model is trained with an MSE loss on the subsequent time step. In early exploration runs, we found $k = 1$ input steps to perform best for this direct next-step prediction setup with *U-Net* (and similarly for *ResNet* and *FNO* below), when investigating $k \in \{1, 2, 3, 4\}$. However, compared to the difference between architectures, these changes are minor.

The additional *U-Net* variants with time unrolling during training share the same architecture. They are likewise trained with an MSE loss applied equally to every step of the predicted rollout with length m against the ground truth. A *U-Net* trained with, e.g., $m = 8$ is denoted by *U-Net*$_{m8}$ below. To keep a consistent memory level during

training, the batch size is reduced correspondingly when m is increased. Thus, the training time of *U-Net* significantly depends on m. While $m = 2$ allows for a batch size of 64, $m = 4$ reduces that to 32, $m = 4$ leads to 16, and finally, for $m = 16$ the batch size is only 8.

We also analyze *U-Net* variants with training noise to stabilize predictions (Sanchez-Gonzalez et al., 2020). Normally distributed noise with standard deviation n is added to every model input during training, while leaving the prediction target untouched. At inference time, the models operate identically to their counterparts without training noise. In the following, *U-Net* models trained with training noise of e.g., $n = 10^{-1}$, are denoted by *U-Net*$_{n1e\text{-}1}$. All *U-Net* variants were trained for 1000 epochs on `Inc` and `Tra`, and 100 epochs on `Iso`.

B.3. PDE-Refiner Implementation

PDE-Refiner is a recently proposed multi-step refinement process to improve the stability of learned PDE predictions (Lippe et al., 2023). This approach relies on starting from the predictions of a trained one-step model, and iteratively refining them by adding noise of decreasing variances and denoising the result. The resulting model is then autoregressively unrolled to form a prediction trajectory. This method implies, that only probabilistic refinements are applied to a deterministic initial prediction. To train a model that can predict and refine at the same time, a random step $r \in [0, R]$ in the refinement process is sampled, and the model is trained with a next-step MSE objective if $r = R$ and with a standard denoising objective otherwise[4]. We re-implement this method, closely following the provided pseudocode in their paper, only changing the backbone network to our *U-Net* implementation (see App. B.2) for a fair comparison against our architectures. The resulting models are trained for 3000 epochs on `Inc` and `Tra`, and 100 epochs on `Iso`, with a batch size of 64 and $k = 1$ input steps.

The authors report that *Refiner* models with around $R = 4$ refinement steps perform best, when paired with a custom, exponential noise schedule, parameterized with a minimum noise variance[5] around $\sigma = 10^{-6}$. As such, we use these values in the main paper (only changing $\sigma = 10^{-5}$ for `Iso`). Below, we additionally sweep over combinations of $R \in \{2, 4, 8\}$ and $\sigma \in \{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$, to investigate the stability with respect to these hyperparameters in our setting. We denote models trained with e.g., $R = 2$ and $\sigma = 10^{-3}$ by *Refiner*$_{R2,\sigma 1e\text{-}3}$ in the following.

[4]Compared to Lippe et al. (2023), we switch the notation to R being the first step in the reverse process here, in line with our notation above, which also matches the notation in the original DDPM (Ho et al., 2020).

[5]For brevity, we use σ for the minimum noise variance here, Lippe et al. (2023) refer to it as σ^2_{min}.

B.4. Implementation of dilated ResNets

For the implementation of *ResNet*$_{dil.}$ and *ResNet*, we follow the setup proposed in (Stachenfeld et al., 2022) that relies on a relatively simple architecture: both models consist of 4 blocks connected with skip connections as originally proposed in (He et al., 2016). Furthermore, one encoder layer before and one decoder convolution layer after the blocks are used to achieve the desired number of input and output channels. Each block contains 7 convolution layers with kernel size 3, stride 1, and 144 feature channels, followed by ReLU activations. For the *ResNet*$_{dil.}$ model, the convolution layers in each block employ the following dilation and padding values: $(1, 2, 4, 8, 4, 2, 1)$. For *ResNet*, all dilation and padding values are set to 1. Both models use a batch size of 64, receive $k = 1$ input steps, predict a single next step, and are trained via an mean-squared-error (MSE) on the prediction against the simulation trajectory as described in App. B.2.

B.5. Implementation of FNOs

For the implementation of the *FNO* variants, we follow the official PyTorch FNO implementation.[6] The lifting and projection block setups are directly replicated from Li et al. (2021), and all models use 4 FNO layers. We vary the number of modes that are kept in in x- and y-direction in each layer as follows: *FNO*$_{16}$ uses $(16, 8)$ modes and *FNO*$_{32}$ uses $(32, 16)$ modes. To ensure a fair comparison, the hidden size of all models are parameterized to reach a number of trainable parameters similar to *ACDM*, i.e. 112 for *FNO*$_{16}$ and 56 for *FNO*$_{32}$. Both models use a batch size of 64, receive $k = 1$ input steps, predict a single next step, and are trained via an mean-squared-error (MSE) on the prediction against the simulation trajectory as described in App. B.2.

B.6. Latent Transformer Implementation

To adapt the approach from Han et al. (2021) to regular grids instead of graphs, we rely on CNN-based networks to replace their Graph Mesh Reducer (GMR) network for encoding and their Graph Mesh Up-Sampling (GMUS) network for decoding. Our encoder model consists of convolution+ReLU blocks with MaxPools and skip connections. In the following, convolution parameters are given as *"input channels → output channels, kernel size, stride, and padding"*. Pooling parameters are given as *"kernel size, stride"*, and Upsampling parameters are give as *"scale factor in x, scale factor in y, interpolation mode"*. The number of channels of the original flow state are denoted by in, the encoder width is w_e, the decoder width is w_d, and L is the

[6]https://github.com/NeuralOperator/neuraloperator

size of the latent space. The encoder layers are:

1. Conv($in \rightarrow w_e, 11, 4, 5$) + ReLU + MaxPool$(2, 2)$
2. Conv($w_e + in_1 \rightarrow 3 * w_e, 5, 1, 2$) + ReLU + MaxPool$(2, 2)$
3. Conv($3 * w_e + in_2 \rightarrow 6 * w_e, 3, 1, 1$) + ReLU
4. Conv($6 * w_e + in_2 \rightarrow 4 * w_e, 3, 1, 1$) + ReLU
5. Conv($4 * w_e + in_2 \rightarrow w_e, 3, 1, 1$) + ReLU
6. Conv($w_e + in_2 \rightarrow L, 1, 1, 0$) + ReLU + MaxPool$(2, 2)$

Here, in_1 and in_2 are skip connections to spatially reduced inputs that are computed directly on the original encoder input with an AvgPool$(8, 8)$ and AvgPool$(16, 16)$ layer, respectively. Finally, the output from the last convolution layer is spatially reduced to a size of 1 via an adaptive average pooling operation. This results in a latent space with L elements. This latent space is then decoded with the following decoder model based on convolution+ReLU blocks with Upsampling layers:

1. Conv($L \rightarrow w_d, 1, 1, 0$) + ReLU + Up$(4, 2, \textit{nearest})$
2. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Up$(2, 2, \textit{nearest})$
3. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Up$(2, 2, \textit{nearest})$
4. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Up$(2, 2, \textit{nearest})$
5. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Up$(2, 2, \textit{nearest})$
6. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Up$(2, 2, \textit{bilinear})$
7. Conv($w_d + L \rightarrow w_d, 5, 1, 2$) + ReLU
8. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU
9. Conv($w_d \rightarrow in, 3, 1, 1$)

Here, the latent space is concatenated along the channel dimension and spatially expanded to match the corresponding spatial input size of each layer for the skip connections. In our implementation, an encoder width of $w_e = 32$, a decoder width of $w_d = 96$ with a latent space dimensionality of $L = 32$ worked best across experiments. For the model *TF-Enc* on the experiments `Inc` and `Tra`, we employ $L = 31$ and concatenate the scalar simulation parameter that is used for conditioning, i.e., Reynolds number for `Inc` and Mach number for `Tra`, to every instance of the latent space. For *TF-VAE* we proceed identically, but here every latent space element consists of two network weights for mean and variance via reparameterization as detailed in (Kingma & Welling, 2014). For *TF-MGN*, we use an additional first latent space of size L that contains a simulation parameter encoding via an MLP as proposed in (Han et al., 2021). Compared to our improved approach, this means *TF-MGN* is not capable to change this quantity over the course of the simulation.

For the latent processor in *TF-MGN* we directly follow the original transformer specifications in (Han et al., 2021) via a single transformer decoder layer with four attention heads and a layer width of 1024. Latent predictions are learned as a residual from the previous step. For our adaptations *TF-Enc* and *TF-VAE*, we instead use a single transformer encoder layer and learn a full new latent state instead of a residual prediction.

To train the different transformer variants end-to-end, we always use a batch size of 8. We train each model with a training rollout of $m = 60$ steps ($m = 50$ for `Iso`) using a transformer input window of $k = 30$ steps ($k = 25$ for `Iso`). We first only optimize the encoder and decoder to obtain a reasonably stable latent space, and then the training rollout is linearly increased step by step as proposed in (Han et al., 2021). We start increasing the rollout at epoch 300 (40 for `Iso`) until the full sequence length is reached at epoch 1200 (160 for `Iso`). Each model is trained with an MSE loss over the full sequence (adjusted to the current rollout length). On `Inc` and `Tra` these transformer-based models were trained for 5000 epochs, and on `Iso` for 200 epochs.

We do not train the decoder to recover values inside the cylinder area for `Inc` and `Tra`, by applying a binary masking (also see App. A.1 for details) before the training loss computation. Note that this masking is not suitable for autoregressive approaches in the input space, as the masking can cause a distribution shift via unexpected values in the masked area during inference, leading to instabilities. The pure reconstruction, i.e. the first step of the sequence that is not processed by the latent processor, receives a relative weight of 1.0, and all steps of the rollout *jointly* receive a weight of 1.0 as well, to ensure that the model balances reconstruction and prediction quality. For *TF-VAE*, an additional regularization via a Kullback–Leibler divergence on the latent space with a relative weight of 0.1 is used. As detailed in (Kingma & Welling, 2014), for a given mean l_m^i and log variance l_v^i of each latent variable l^i with $i \in 0, 1, \ldots, L$, the regularization $\mathcal{L}_{KL}$ is computed as

$$\mathcal{L}_{KL} = -0.5 * \frac{1}{L} * \sum_{i=0}^{L} 1 + l_v^i - {l_m^i}^2 - e^{l_v^i}.$$

C. Training and Inference Performance

All model architectures were trained, evaluated, and benchmarked on a server with an NVIDIA RTX A5000 GPU with 24GB of video memory and an Intel Xeon Gold 6242R CPU with 20 cores at 3.1 GHz. A performance overview across models can be found in Tab. 3. The training time column indicates how many hours are approximately required to fully train a single model according to the epochs and batch size given further left. For each architecture, we train 3

Table 3. Overview of training and inference performance for different model architectures.

Architecture	Training Epochs Inc / Tra / Iso	Batch Size	Training Time [h] Inc / Tra / Iso	Training Speed per Epoch [min] Inc / Tra / Iso	Inference Speed [s] without I/O	Inference Speed [s] with I/O
$ACDM_{R20}$	3100 / 3100 / 100	64	66 / 42 / 62	0.80 / 0.74 / 37.4	193.9	195.7
$ACDM_{R100}$					973.2	975.0
U-Net	1000 / 1000 / 100	64	26 / 19 / 90	1.16 / 1.10 / 55.0	9.4	11.1
$U\text{-}Net_{m4}$		32	34 / 31 / 156	1.97 / 1.82 / 95.3		
$U\text{-}Net_{m8}$		16	45 / 43 / 217	2.67 / 2.49 / 130.8		
$U\text{-}Net_{m16}$		8	54 / 51 / 262	3.20 / 3.03 / 161.1		
TF_{MGN}	5000 / 5000 / 200	8	43 / 42 / 69	0.51 / 0.48 / 20.8	0.8	2.8
TF_{Enc}			38 / 38 / 67	0.42 / 0.43 / 19.9	0.6	2.8
TF_{VAE}			38 / 38 / 68	0.42 / 0.42 / 20.0	0.7	2.7
$ResNet_{dil.}$	1000 / 1000 / 100	64	52 / 49 / 263	3.15 / 2.95 / 152.5	4.2	6.0
ResNet						
FNO_{16}	2000 / 2000 / 200	64	13 / 12 / 55	0.36 / 0.34 / 16.5	2.3	4.1
FNO_{32}			8 / 8 / 33	0.22 / 0.21 / 9.9	2.5	4.2
$Refiner_{R2}$	3000 / 3000 / 100	64	59 / 55 / 61	1.18 / 1.10 / 37.2	30.5	32.8
$Refiner_{R4}$					60.7	63.2
$Refiner_{R8}$					92.7	94.4

models (2 for Iso) based on randomly seeded runs for the evaluations in the paper, and report the maximum training time. The training speed per epoch in minutes in the next column is averaged over a set of training epochs across all trained models.

All model architectures were trained on each data set until their training loss curves were visually fully converged. This means, architectures with more complex learning objectives require more epochs compared to simpler methods. As such, the transformer variants are highly demanding, as they first need to learn a good latent embedding via the encoder and decoder, and afterwards need to learn the transformer unrolling schedule that is faded in during training time. *ACDM* which needs to learn a full denoising schedule via random sampling can also require more training iterations compared to direct next-step predictors such as *U-Net*, *ResNet*, or *FNO*. Furthermore, we found the performance of next-step predictors to degrade when trained substantially past the point of visual convergence in early exploration runs. As mentioned above, the default training batch size of 64 is reduced for architectures that exceed available GPU memory, so the training time comparison is performed at roughly equal memory. Thus, training unrolled *U-Net* models is highly expensive, especially on Iso, both via higher memory requirements that result in a lower batch size, but also in the number of computations required for the training rollout.

The right side of Tab. 3 features the inference speed of each method. It is measured on a single example sequence consisting of $T = 1000$ time steps. We report the overall time in seconds that each architecture required during inference for this sequence. Shown in the table is the pure model inference time, as well as the performance including I/O operations and data transfers from CPU to GPU. Note that compared to the performance of *U-Net*, the inference speed slowdown factor of the diffusion-based architecture *Refiner* and *ACDM* is closely related to the number of refinement or diffusion steps R. This directly corresponds to the number of backbone model evaluations.

D. Accuracy Evaluation with LSiM

The LSiM metric (Kohl et al., 2020) is a deep learning-based similarity measure for data from numerical simulations. It is designed to more accurately capture the similarity behavior of larger patterns or connected structures that are neglected by the element-wise nature of point-based metrics like MSE. As a simple example, consider a vortex inside a fluid flow that is structurally correctly predicted, but spatially misplaced compared to a reference simulation. While MSE would result in a large distance value, LSiM results in a relatively low distance, especially compared to another vortex that is spatially correctly positioned, but structurally different. LSiM works by embedding both inputs that should be compared in a latent space of a feature extractor network, computing an element-wise difference, and aggregating this difference to a scalar distance value via different operations. The metric is trained on a range of data sets consisting of different transport-based PDE simulations like advection-diffusion equations, Burgers' equation, or the full Navier-Stokes equations. It has been shown to generalize well to flow simulation data outside its training domain like isotropic turbulence.

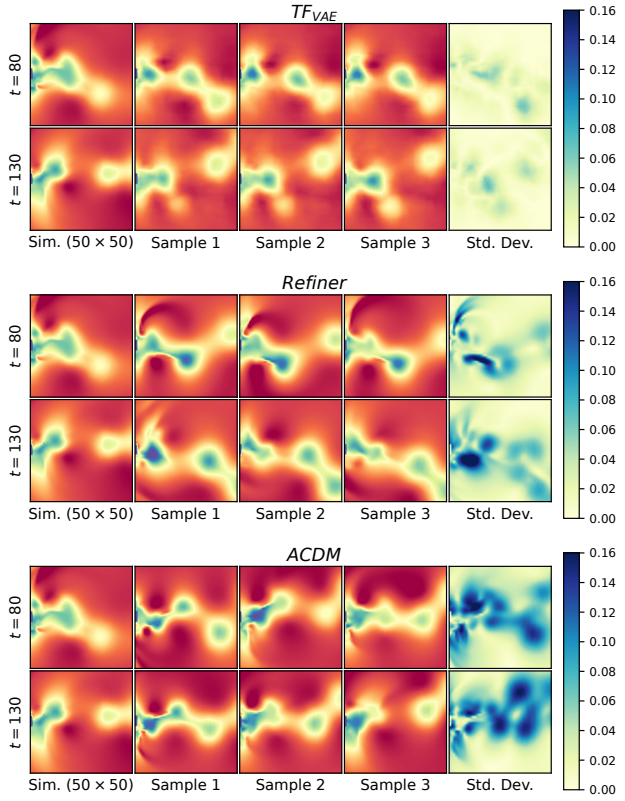

Figure 7. Large-scale posterior sample comparison with standard deviation on $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$ (pressure) at different time steps t (also see accompanying posterior sampling videos).

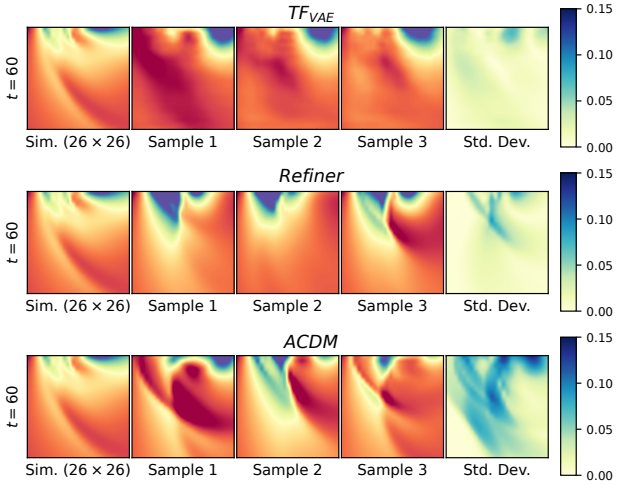

Figure 8. Small-scale posterior sample comparison with according standard deviation on a sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$ (pressure) at different time steps t.

E. Posterior Sampling Analysis

One of the most attractive aspects of a DDPM-based simulator is posterior sampling, i.e., the ability to create different samples from the solution manifold for one initial condition. Below, we qualitatively and quantitatively evaluate the posterior samples of the investigated probabilistic methods *TF$_{VAE}$*, *Refiner*, and *ACDM*. For this purpose, we focus on the transonic flow experiment as a representative case with medium difficulty. For `Inc`, it is visually difficult to discern predictions due the simpler learning task, while it can be difficult to judge the larger discrepancies from the simulation reference that can occur for the underdetermined `Iso` experiment.

Qualitative Analysis Figures 7 and 8 visualize random, exemplary posterior samples from the different methods at different spatial zoom levels, alongside the snapshot of the corresponding $\texttt{Tra}_{\texttt{long}}$ reference simulation. Furthermore, the spatially varying standard deviations across the five computed samples from each method are included. First, it becomes apparent that *TF$_{VAE}$* achieves barely any visual or physical differences across samples, as the general vortex structure downstream of the cylinder remain highly similar, even far into the trajectory. *Refiner* fares better and creates differences in the predictions, especially for large t. However, as clearly visible in Fig. 8, both approaches struggle to create physically important, small-scale details with high frequencies, such as the strongly varying formation of shock waves near the immersed cylinder. The predictions from *TF$_{VAE}$* lack these features entirely, while *Refiner* is sometime able to create them, however in a quite similar and sometimes unphysical manner. The diffusion approach produces the most realistic, and diverse features, while even being able to recreate physically plausible shock wave configurations. As expected, the spatial standard deviation increases over time due to the chaotic nature of this test case. The locations of high variance for *ACDM* match areas that are more difficult to predict, such as vortices and shock wave regions.

Quantitative Analysis To analyze the quality of a distribution of predicted simulation trajectories from a probabilistic algorithm, it is naturally not sufficient to directly compare to a single target sequence, as even highly accurate numerical simulations would eventually decorrelate from a target simulation over time (Hu & Liao, 2020). Instead, our experimental setup allows for using temporal and spatial evaluations to measure whether different samples *statistically* match the reference simulation, as established by turbulence research (Dryden, 1943). Two metrics for a sequence from $\texttt{Tra}_{\texttt{long}}$ are analyzed here: We evaluate the wavenumber of the horizontal motion across a vertical line in the flow (averaged over time), and the temporal frequency of the vertical motion at a point probe. The mean and variance across trained models and samples of the resulting spectra are shown in Fig. 9. While similar at first glance, some key differences between the spectra of the different architectures

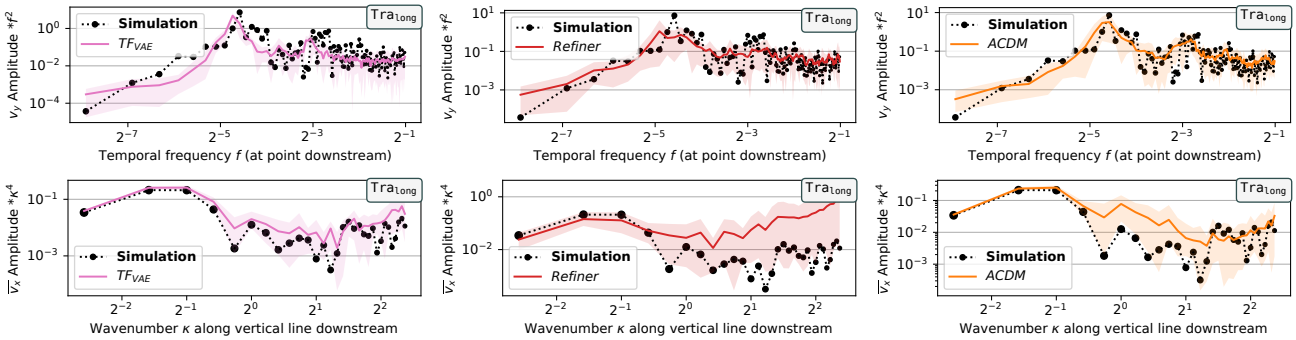

Figure 9. Temporal (top) and spatial (bottom) frequency analysis across posterior samples for a full sequence from `Tra`$_{\texttt{long}}$ with $Ma = 0.65$. The shaded area shows the 5$^{\text{th}}$ to 95$^{\text{th}}$ percentile across all trained models and posterior samples.

can be observed. For the temporal analysis, both *TF$_{VAE}$* and *Refiner* fail to accurately reproduce the main vortex shedding frequency indicated by the peak around a frequency of 2^{-5}, while *ACDM* comes very close to the reference. The high frequency content on the right side of the spectrum is on average also best reproduced by *ACDM*, as for example visible by the minor peak around a frequency of 2^{-3}. In terms of the spatial spectra at the bottom of Fig. 9, *Refiner* fails to capture the behavior of the reference simulation. The most important low spatial frequencies have high variance, and the amplitudes for medium and high frequencies do not match the reference entirely. *TF$_{VAE}$* and *ACDM* result in much better spatial spectra, and both accurately capture the major low frequencies. *ACDM* overshoots in the medium frequency regime while *TF$_{VAE}$* has minor discrepancies for high frequencies. Overall, the samples from *ACDM* statically most accurately reflect the physical behavior of the reference simulation.

F. Comparing Spectral Statistics

Spectral statistics also highlight the differences between the other model architectures under consideration. On `Iso`, the temporal frequency of the x-velocity is evaluated and shown at the top of Fig. 10. As this case is isotropic, we average the evaluation across every spatial point for a more stable analysis. Furthermore, we also analyze the spatial behavior on `Iso` via an energy spectrum at the bottom of Fig. 10. In that case, the turbulent kinetic energy (TKE) is evaluated and aggregated in x- and y-direction, and averaged across all time steps of the simulation. *TF$_{Enc}$* is lacking on the spatial and temporal frequency band, and *TF$_{MGN}$* and *TF$_{VAE}$* which are not shown behave similarly. This behavior is caused by the compression to the latent space, where spatial details are lost, and meaningful temporal evolution is challenging due to the overall complexity of the `Iso` experiment.

Architectures which operate as direct next-step predictors such as *ResNet$_{dil.}$*, *FNO$_{16}$*, *U-Net*, and *ACDM$_{ncn}$* clearly overshoot, due to the direct error propagation that causes

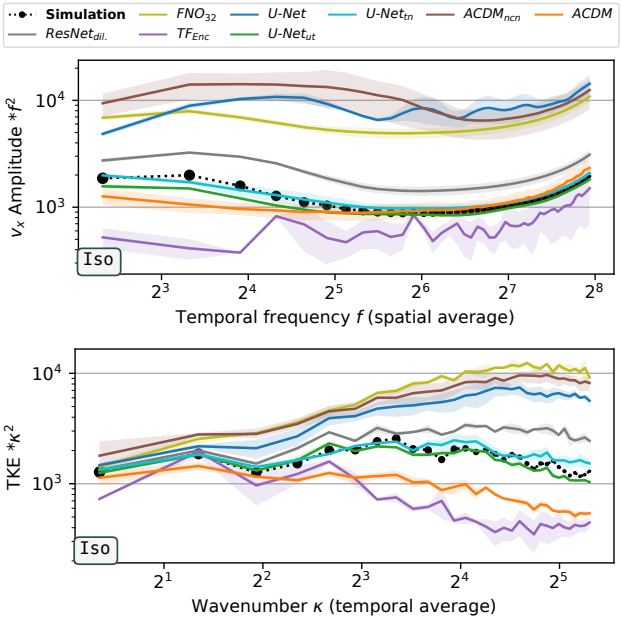

Figure 10. Temporal (top) and spatial (bottom) frequency analysis on a sequence from `Iso` with $z = 300$. The shaded area shows the 5$^{\text{th}}$ to 95$^{\text{th}}$ percentile across all trained models and posterior samples. PDE-Refiner is omitted here, as some of its models and samples are unstable, leading to substantially worse results compared to all other methods on both evaluations.

instabilities. Introducing explicit stabilization techniques via unrolling or training noise to *U-Net* substantially improves spatial and temporal spectral behavior, making them the most accurate architectures in this evaluation. However, early signs of instabilities can be observed for *U-Net$_{tn}$*, as additional energy is introduced in high spatial frequencies compared to the reference which will eventually result in instabilities. *ACDM* follows close behind the stabilized *U-Net* variants: High temporal and low spatial frequencies are modeled well, but it deviates in terms of lower temporal and higher spatial frequencies. This is most likely caused by the strongly under-determined nature of `Iso`. It causes *ACDM* to unnecessarily dissipate spatial high-frequency mo-

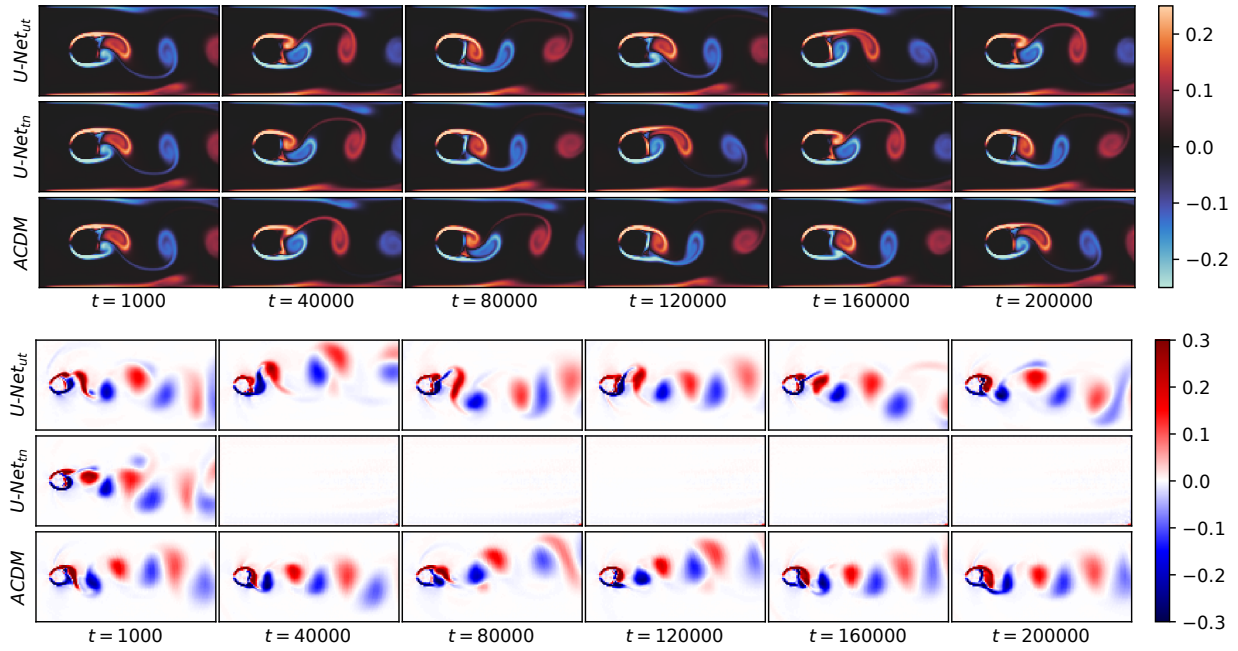

Figure 11. Predictions (vorticity) from the most temporally stable architectures, when unrolled for $T = 200\,000$ steps on a sequence from $\texttt{Inc}_{\texttt{high}}$ with $Re = 1000$ (top), and on a sequence from $\texttt{Tra}_{\texttt{ext}}$ with $Ma = 0.52$ (bottom).

tions, which in turn impacts low temporal frequencies over longer rollouts. *Refiner* is omitted in both visualizations, as several posterior samples across training runs were unstable, causing substantially worse result compared to all other methods.

G. Stability on Extremely Long Rollouts

To further investigate the temporal stability of the most stable methods *U-Net*$_{ut}$, *U-Net*$_{tn}$, and *ACDM* in our comparison, we provide results on extremely long inference rollouts. We choose sequences from the extrapolation areas of our data sets for this purpose: three sequences from $\texttt{Inc}_{\texttt{high}}$ with $Re \in \{960, 980, 1000\}$ and three from $\texttt{Tra}_{\texttt{ext}}$ with $Ma \in \{0.50, 0.51, 0.52\}$. Every architecture is unrolled over $T = 200\,000$ steps, for the three training runs each. Figure 11 visualizes the resulting predictions. All methods and training runs remain fully stable over the entire rollout on the sequence from $\texttt{Inc}_{\texttt{high}}$. Since this case is unsteady but fully periodic, the results of all models is a simple, periodic trajectory that prevents error accumulation. For the sequences from $\texttt{Tra}_{\texttt{ext}}$, one from the three trained *U-Net*$_{tn}$ models has stability issues within the first few thousand steps and deteriorates to a simple, mean flow prediction without vortices. The remaining training runs do however remain stable. *U-Net*$_{ut}$ and *ACDM* on the other hand are fully stable across training runs for this case, indicating a fundamentally higher resistance to rollout errors which eventually could cause instabilities. Due to its highly chaotic nature, relying on simple, periodic predictions is not sufficient for this case: As displayed in Fig. 11, the predictions of *U-Net*$_{ut}$, but especially *ACDM*, exhibit differences at the same stage of the vortex shedding period. This is most clearly visible when comparing the predictions for $t = 120\,000$, $t = 160\,000$, and $t = 200\,000$, where a blue vortex facing downwards is about to detach behind the cylinder.

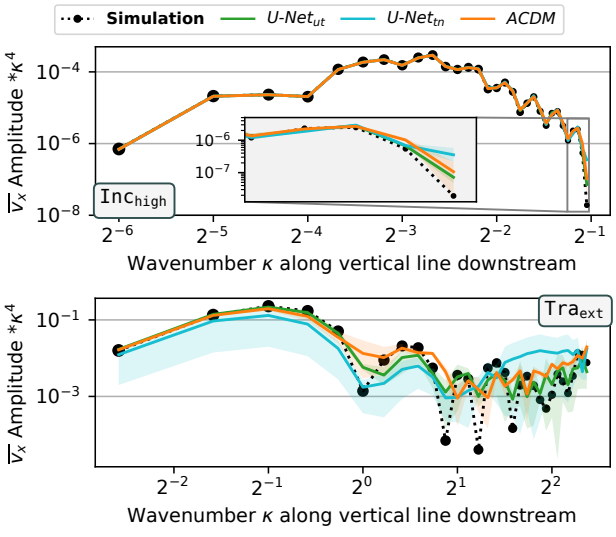

Figure 12. Spatial frequency along a vertical line downstream for a sequence from $\texttt{Inc}_{\texttt{high}}$ with $Re = 1000$ (top), and on a sequence from $\texttt{Tra}_{\texttt{ext}}$ with $Ma = 0.50$ (bottom). The prediction spectra are computed on the mean flow achieved from averaging every 100^{th} step from a prediction with a horizon of $T = 200\,000$ steps.

In Fig. 12, the statistical match of these long-term predictions to the physical behavior of the simulation trajectories is analyzed. We evaluate the spatial frequency spectrum of the mean horizontal flow velocity along a vertical line downstream of the cylinder. For the predictions, every 100$^{\text{th}}$ step of the predicted rollout from every training run is used to compute the mean flow. Note that the corresponding simulation spectrum is only computed over the simulated time range of $t \in [300, 1300)$ for $\texttt{Inc}_{\texttt{high}}$, and $t \in [0, 1000)$ for $\texttt{Tra}_{\texttt{ext}}$ with temporal strides of 2 leading to $T = 500$ time steps for this evaluation. For the rather simple sequence from $\texttt{Inc}_{\texttt{high}}$, all methods perfectly match the simulation spectrum. For the sequence from $\texttt{Tra}_{\texttt{ext}}$, the spectrum from *U-Net*$_{tn}$ has a high standard deviation across the frequency band due the diverging training run. *U-Net*$_{ut}$ and *ACDM* statistically match the low simulation frequencies here very well, and only exhibit minor deviations for the higher frequencies, indicating that the predictions do not drift substantially over extremely long rollout horizons.

G.1. Stability Criteria for Unrolled Training

For the *U-Net* models with unrolled training we also investigated key criteria to achieve fully stable rollouts over extremely long horizons. For this purpose, different ablation architectures are evaluated on the long rollout experiments over $T = 200\,000$ rollout steps described above. Figure 14 displays the percentage of stable runs across architectures for three trained models for every sequence from $\texttt{Tra}_{\texttt{ext}}$, as well as an average stability. As shown above *ACDM* remains fully stable while only two out of three *U-Net*$_{tn}$ are stable. The most important stability criterion for *U-Net* trained with unrolling is the number of unrolling steps m: while models with $m \leq 4$ do not achieve stable rollouts, using $m \geq 8$ is sufficient for stability across Mach numbers.

Three factors that did not substantially impact rollout stability in our experiments are (***i***) the prediction strategy, (***ii***) the amount of training data, and (***iii***) the backbone architecture. First, using residual predictions, i.e., $s^t = s^{t-1} + f_\theta(s^{t-1})$ instead of $s^t = f_\theta(s^{t-1})$ does not impact stability for different values of m as shown in the middle of Fig. 14. Second, the stability is not affected when reducing the amount of available training data by a factor of 8 from 1000 steps per Mach number to 125 steps. These models are also trained with $8\times$ more epochs to ensure a fair comparison. This training data reduction still retains the full physical behavior, i.e., complete vortex shedding periods. Third, it possible to train other backbone architectures with unrolling to achieve fully stable rollouts, as shown on the right in Fig. 14: *ResNet*$_{dil.}$ models trained with $m = 8$ are also able to keep predictions stable across the entire prediction horizon. For *ResNet* only one trained model is stable, most likely due to the reduced receptive field. However, we expect achieving full stability is also possible with longer training rollout horizons.

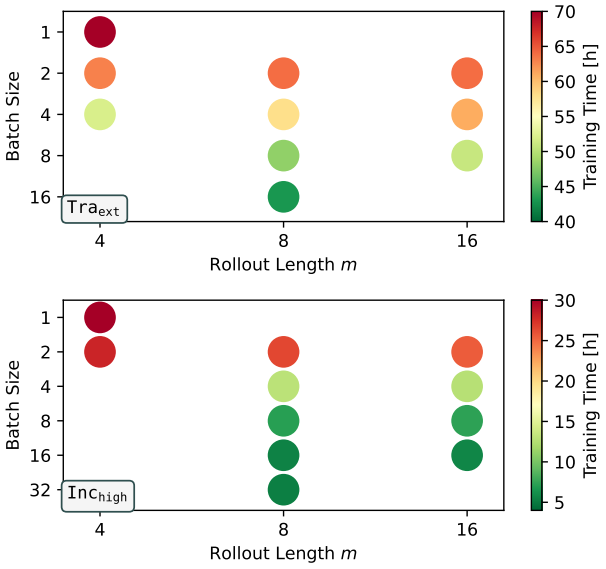

Figure 13. Training time for different combinations of rollout length m and batch size on $\texttt{Tra}_{\texttt{ext}}$ (top) and $\texttt{Inc}_{\texttt{high}}$ (bottom). Only configurations that result in highly stable rollouts are shown (percentage of stable runs across three trained models and three sequences $\geq 89\%$). Note that the training time increases faster for smaller batches compared to longer rollouts.

Finally, we observed that the batch size can impact the stability of models trained with unrolling. In the image domain, it has been documented that smaller batch sizes exhibit better generalization properties compared to larger mini-batch sizes or even full gradient decent without mini-batches (Goodfellow et al., 2016). As described in more detail in App. B.2, we adjust the batch size of *U-Net* for different values of m, such that each batch contains the same amount of data, e.g., halving the batch size when doubling m. By default, we chose the largest possible batch size that fits in GPU memory in the remainder of this work, to ensure computational efficiency. Here, we investigate different configurations by varying of batch size and rollout length on $\texttt{Tra}_{\texttt{ext}}$ and $\texttt{Inc}_{\texttt{high}}$ in Figs. 15 and 16, respectively. Training models with smaller batches for the same amount of network updates did not improve stability, so all networks are trained with the same amount of data, i.e., an equal number of epochs with more network updates for small batch sizes. Note that the width of the *U-Net* architecture was substantially reduced by a factor of 8 across all network layers for $\texttt{Inc}_{\texttt{high}}$ in Fig. 16 to artificially increase the difficulty of the learning task, as otherwise every model configuration would be fully stable.

For both test sets, *U-Net*$_{m4}$ with the largest batch size that fits in memory is not stable, while perfect stability can be achieved with lowering the batch size. However, this effect can not fully replace the stabilization from longer training rollouts, as *U-Net*$_{m2}$ is never stable. Furthermore,

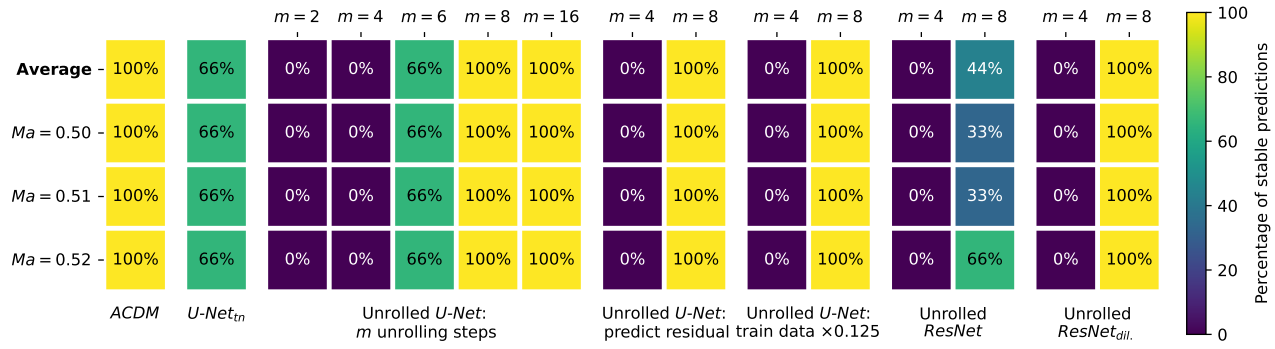

Figure 14. Important aspects to achieve stable models for extremely long rollouts on $\texttt{Tra}_{\texttt{ext}}$. Shown is the percentage of stable runs for three model seeds across sequences with $Ma \in \{0.50, 0.51, 0.52\}$ and their average. While *ACDM* is fully stable out-of-the-box, one *U-Net*$_{tn}$ model diverges across Mach numbers. The most important aspect for fully stable unrolled *U-Net* models is the rollout length m, as displayed in the third block. However, the training methodology of predicting residuals instead of full next states did not impact the stability. Similarly, reducing the training data by a factor of 8 (with an $8\times$ longer training) did not alter the results across values of m. Other architectures like *ResNet* and *ResNet*$_{dil.}$ are also capable to achieve full stability when trained with unrolling, even though *ResNet* requires slightly larger m for consistent stability.

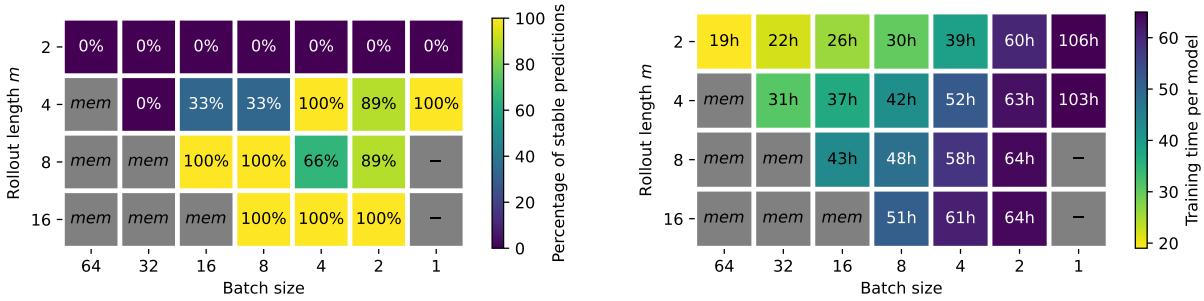

Figure 15. Stability investigation for unrolled *U-Net* models with different combinations of batch size and rollout length m on $\texttt{Tra}_{\texttt{ext}}$. Shown are the percentage of stable predictions across three models and three sequences with $T = 200\,000$ steps each (left) and approximate training time for each parameter combination (right). Grey configurations are infeasible due to memory constraints (*mem*) or omitted due to high computational resource demands (–). Notice that decreases in batch size can lead to more stable models for medium m, but incur a higher training cost compared to increasing m.

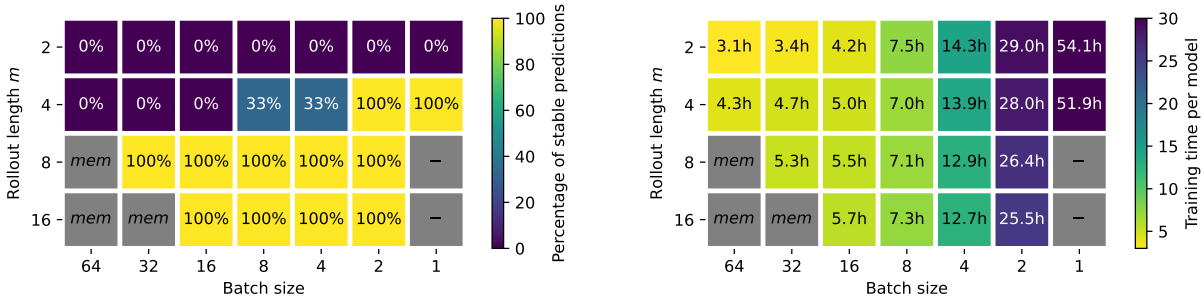

Figure 16. Stability investigation for unrolled *U-Net* models (with a width reduced by a factor of 8) with different combinations of rollout length m and batch size on $\texttt{Inc}_{\texttt{high}}$. Shown are the percentage of stable predictions across three sequences with $T = 200\,000$ steps and three models each (left) and approximate training time for each configurations (right). Grey parameter combinations are infeasible due to memory constraints (*mem*) or omitted due to high computational resource demands (–). Similar to Fig. 15 decreasing the batch size can lead to higher stability models for medium m, but this affects training cost. Note that there is barely any overhead for additional unrolling steps, as the model is small and longer training rollouts are implemented via fewer training sequences.

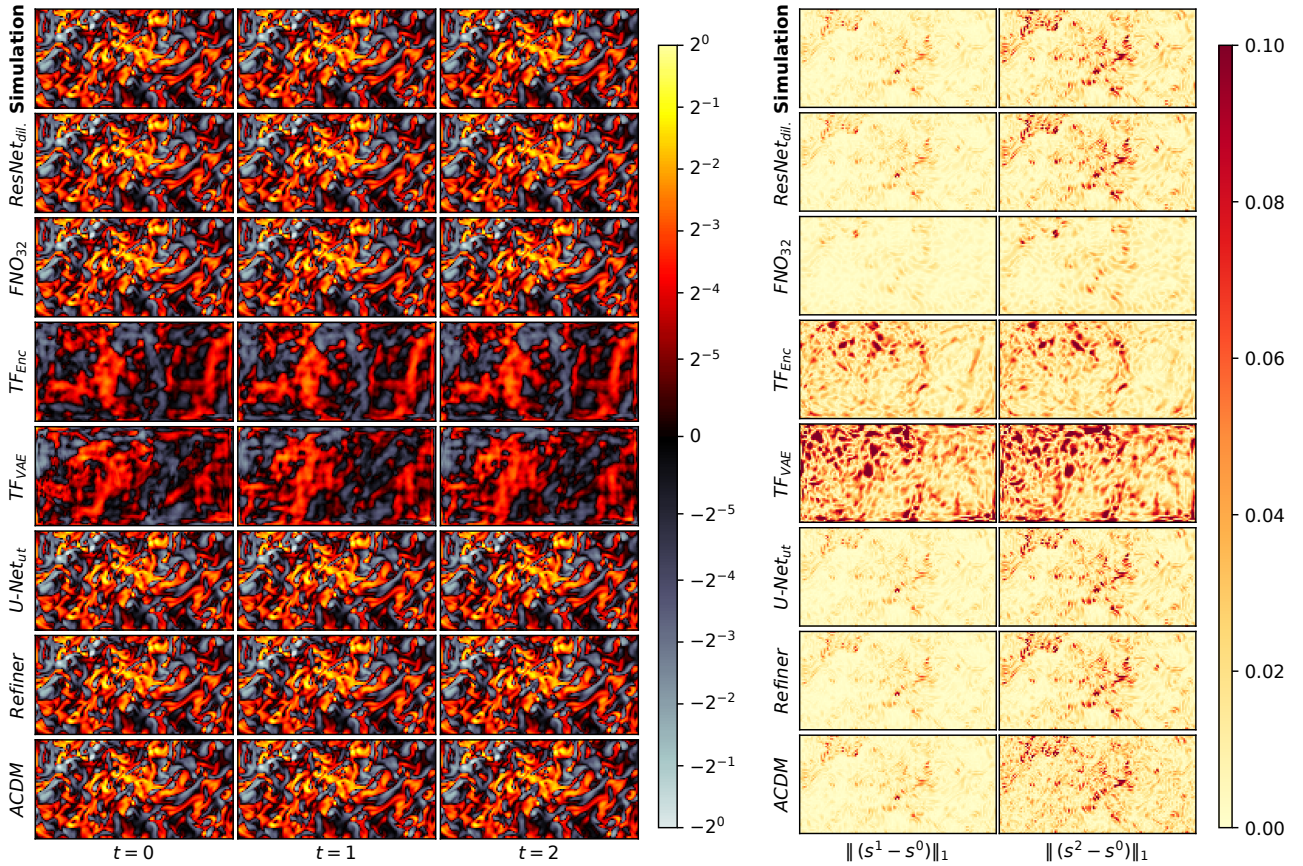

Figure 17. Temporal coherence analysis of different model architectures. Model predictions (left) and the differences between the first and following two prediction steps (right) are shown an example from `Iso` with $z = 300$ (also see accompanying temporal coherence videos).

smaller batch sizes are less memory efficient meaning model training takes longer. Since we implement longer training rollouts via fewer training sequences, larger values of m do not necessarily induce higher training times as shown for the small *U-Net* on $\texttt{Inc}_{\texttt{high}}$ in Fig. 16. Nevertheless, larger training rollouts did lead to longer training times for the full-size *U-Net* model as shown in Fig. 15 and also listed in Tab. 3. Thus, we investigate the most efficient combination of rollout length and batch size to achieve fully stable rollouts. As shown in Fig. 13, choosing longer rollouts over smaller batches leads to high stability at lower training cost, for both $\texttt{Inc}_{\texttt{high}}$ and $\texttt{Tra}_{\texttt{ext}}$. However, the difference did change depending on the model size in our experiments. To summarize, the most important criterion for full stable models is the training rollout length. While lowering the batch size did have an effect, choosing large batch sizes was superior in terms of training time.

H. Evaluating Temporal Coherence

Here, we analyze the temporal coherence between individual time steps of the architectures under consideration. In Fig. 17 on the left, we display the first three simulation steps of a sequence from `Iso`, along with the corresponding predictions of models from each architecture class. In addition, the spatial changes between the first two predicted steps and s^0 are visualized on the right. Most architectures do not exhibit issues with temporal coherence and closely follow to difference pattern produced by the simulation, except from the transformer variants: both, *TF$_{Enc}$* or *TF$_{VAE}$*, struggle to reproduce the original vorticity field at $t = 0$, and furthermore show large difference between prediction steps. The key difference for these architectures is, that the decoders of *TF$_{Enc}$* or *TF$_{VAE}$* do not have access to previously generated time steps, as their input is only a sample from the latent space at every step. This leads to temporal artifacts where large differences between consecutive time steps can occur. Note that this problem is in general worse for *TF$_{VAE}$* indicated by the larger overall difference magnitude, as the probabilistic nature of the model makes temporal coherence even more challenging. *ACDM* with $R = 100$ does show some coherence issues in the vorticity, indicated by some very small, slightly darker areas in the difference field, especially for s^2 (see accompanying videos). However, they are quite minor, only visible in the vorticity and not in the raw predicted velocities, and can be further mitigated with additional diffusion steps.

Table 4. Accuracy ablation for different diffusion steps R.

		$\texttt{Tra}_{\texttt{ext}}$		$\texttt{Tra}_{\texttt{int}}$		$\texttt{Iso}$	
Method	R	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-2})	LSiM (10^{-1})
ACDM	10	3.8 ± 1.4	1.8 ± 0.3	6.2 ± 2.5	2.1 ± 0.6	15.1 ± 7.4	6.6 ± 1.4
ACDM	15	2.5 ± 1.5	1.4 ± 0.3	2.7 ± 2.0	1.4 ± 0.5	4.8 ± 1.6	4.3 ± 1.0
ACDM	20	2.3 ± 1.4	1.3 ± 0.3	2.7 ± 2.1	1.3 ± 0.6	4.5 ± 1.3	4.1 ± 0.8
ACDM	30	2.5 ± 1.9	1.4 ± 0.4	2.7 ± 2.3	1.3 ± 0.6	4.8 ± 1.9	4.1 ± 0.9
ACDM	50	2.3 ± 1.4	1.3 ± 0.3	2.4 ± 2.1	1.3 ± 0.6	3.4 ± 0.9	3.4 ± 0.7
ACDM	100	2.3 ± 1.3	1.3 ± 0.3	3.1 ± 2.7	1.4 ± 0.6	3.7 ± 0.8	3.3 ± 0.7
ACDM	500	2.5 ± 1.5	1.4 ± 0.4	3.1 ± 2.5	1.4 ± 0.6	3.5 ± 0.9	3.2 ± 0.7

I. Ablations

In the following, we provide various ablation studies on the number of diffusion steps in App. I.1, as well as ablations on the stabilization techniques of longer training rollouts in App. I.2 and training noise in App. I.3. We investigate different loss formulations in App. I.4, and analyze the impact of the recently proposed architecture modernizations for U-Nets in App. I.5. Finally, we provide ablations on the PDE-Refiner method (Lippe et al., 2023) in App. I.6.

I.1. Ablation on Diffusion Steps

In the following, we will investigate the *ACDM* approach with respect to the effect of the number of diffusion steps R in each autoregressive prediction step. We use the adjusted linear variance schedule as discussed in App. B.1, according to the investigated diffusion step R. At training and inference time, models always use R diffusion steps. Prediction examples for this evaluation can be found in Figs. 36 and 37 in App. K.

Accuracy Tab. 4 contains the accuracy, of *ACDM* models with a different number of diffusion steps R. While too few diffusion steps on $\texttt{Tra}$ are detrimental, as visible for $\textit{ACDM}_{R10}$, adding more steps after around $R = 20$ does not improve accuracy. However, on $\texttt{Iso}$ the accuracy of *ACDM* does continue to improve slightly with increased values of R up to our evaluation limit of $R = 500$. We believe this results from the highly underdetermined setting of the $\texttt{Iso}$ experiment. Note that there is a relatively sharp boundary between too few and a sufficient number of steps; in our experiments $15 - 20$ steps on $\texttt{Tra}$ and $50 - 100$ steps on $\texttt{Iso}$.

Temporal Stability In Fig. 18, we evaluate the temporal stability via the magnitude of the rate of change of s, as detailed in the main paper. Here, different behavior for the ablation models with respect to the number of diffusion steps emerges on $\texttt{Tra}_{\texttt{long}}$ and $\texttt{Iso}$. For the former, too little steps, i.e., for $\textit{ACDM}_{R10}$, result in unwanted, high-frequency temporal spikes that are also visible as slightly noisy predictions. For $15 - 20$ diffusion steps, these issues

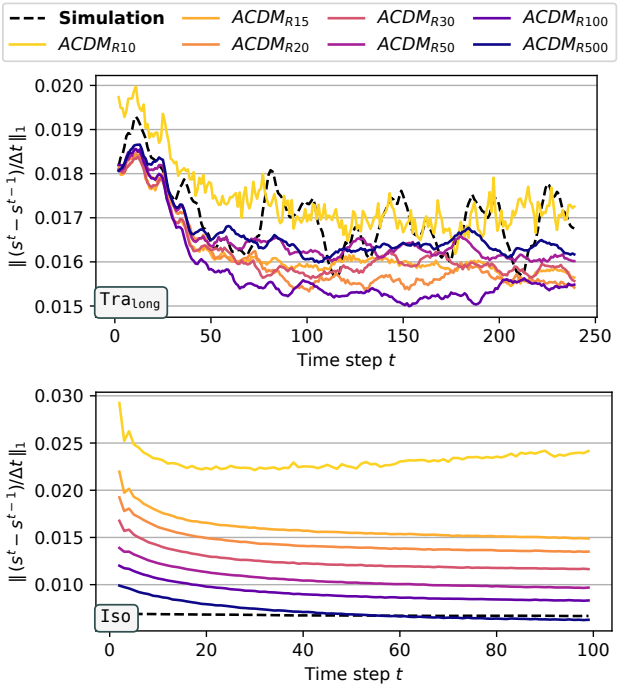

Figure 18. Temporal stability evaluation via error to previous time step for different diffusion steps R on $\texttt{Tra}_{\texttt{long}}$ (top) and $\texttt{Iso}$ (bottom). Standard deviations are omitted for visual clarity.

vanished, and adding further iterations does not substantially improve temporal stability. Only a slightly higher rate of change can be observed for $\textit{ACDM}_{R50}$ and $\textit{ACDM}_{R500}$.

On $\texttt{Iso}$, a tradeoff between prediction accuracy and sampling speed occurs. Even though there are some minor temporal inconsistencies in the first few time steps for very low R, all variants result in a stable prediction. However, the magnitude of the rate of change consistently matches the reference trajectory more closely when increasing R. This also corresponds to a slight reduction in the overly diffusive prediction behavior for large R, both visually and in a spatial spectral analysis via the TKE, as shown in Fig. 19. We believe this tradeoff is caused by the highly underdeter-

Table 5. Accuracy ablation for different training rollout lengths m and pre-training (Pre.).

			$\texttt{Tra}_{\texttt{ext}}$		$\texttt{Tra}_{\texttt{int}}$		$\texttt{Iso}$	
Method	m	Pre.	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-2})	LSiM (10^{-1})
U-Net	2	no	3.1 ± 2.1	3.9 ± 2.8	2.3 ± 2.0	3.3 ± 2.8	25.8 ± 35	11.3 ± 3.9
U-Net	4	no	1.6 ± 1.0	1.4 ± 0.8	1.1 ± 1.0	0.9 ± 0.4	3.7 ± 0.8	2.8 ± 0.5
U-Net	8	no	1.6 ± 0.7	1.1 ± 0.2	1.5 ± 1.5	1.0 ± 0.5	4.5 ± 2.8	2.4 ± 0.5
U-Net	16	no	2.2 ± 1.1	1.3 ± 0.3	2.4 ± 1.3	1.3 ± 0.5	13.0 ± 11	3.8 ± 1.5
U-Net	4	yes	—	—	—	—	5.7 ± 2.6	3.6 ± 0.8
U-Net	8	yes	—	—	—	—	2.6 ± 0.6	2.3 ± 0.5
U-Net	16	yes	—	—	—	—	2.9 ± 1.4	2.3 ± 0.5

mined nature of the Iso experiment, that leads to a weaker conditioned learning setting, that naturally requires more diffusion steps for high-quality results. Furthermore, the predictions of $ACDM_{R100}$ exhibit minor visually visible temporal coherence issues on Iso, where small-scale details can flicker quickly. This is caused by highly underdetermined nature of Iso, and can be mitigate by more diffusion steps as well, as $ACDM_{R500}$ reduces this behavior.

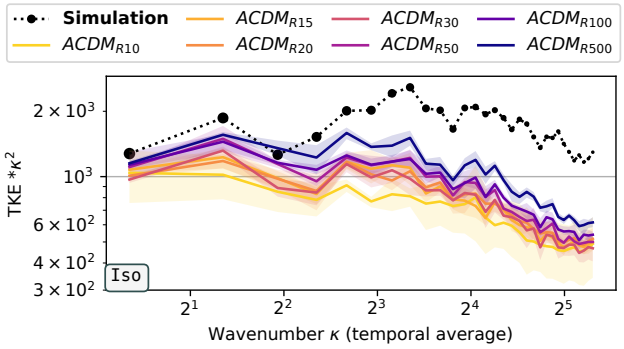

Figure 19. Spatial frequency analysis via the turbulent kinetic energy (TKE) on a sequence from Iso with $z = 300$ for the diffusion step ablation.

Summary *ACDM* works well out-of-the-box with a large number of diffusion steps, but R can be used to balance accuracy and inference performance. Finding the number of diffusion steps for the best tradeoff is dependent on the data set and learning problem formulation. Generally, setups with stronger conditioning work with few diffusion steps, while less restrictive learning problems can benefit from more diffusion samples. In our experiments, the ideal thresholds emerged relatively clearly.

I.2. Ablation on Training Rollout

Here, we investigate the impact of unrolling the *U-Net* model at training time, via varying the training rollout length m. For these models, we use the U-Net architecture as described in App. B.2 with $k = 1$ input steps. However, gradients are propagated through multiple state predictions during training, and corresponding MSE loss over all predicted steps is applied. Prediction examples can be found in Figs. 38 and 39 in App. K.

Accuracy Table 5 shows models trained with different rollout lengths, and also includes the performance of *U-Net* with $m = 2$ for reference. For the transonic flow, $m = 4$ is already sufficient to substantially improve the accuracy compared to *U-Net* for the relatively short rollout of $T = 60$ steps during inference for $\texttt{Tra}_{\texttt{ext}}$ and $\texttt{Tra}_{\texttt{int}}$. Increasing the training rollout further does not lead to additional improvements and only slightly changes the accuracy. However, note that there is still a substantial difference between the temporal stability of $U\text{-}Net_{m4}$ compared to $U\text{-}Net_{m8}$ or $U\text{-}Net_{m16}$ for cases with a longer inference rollout as analyzed below.

On Iso, the behavior of *U-Net* models with longer training rollout is clearly different as models with $m > 4$ substantially degrade compared to $m = 4$. The main reason for this behavior is that gradients from longer rollouts can be less useful for complex data when predictions strongly diverge from the ground truth in early training stages. Thus, we also considered variants, with $m > 2$ that are finetuned from an initialization of a trained basic *U-Net*, denoted by e.g., $U\text{-}Net_{m4,Pre}$. With this pre-training the previous behavior emerges, and $U\text{-}Net_{m8,Pre}$ even clearly improves upon $U\text{-}Net_{m4}$.

Temporal Stability In Fig. 20, we evaluate the temporal stability via the magnitude of the rate of change of s, as detailed in the main paper. On $\texttt{Tra}_{\texttt{long}}$ all models perform similar until about $t = 50$ where *U-Net* deteriorates. $U\text{-}Net_{m4}$ also exhibits similar signs of deterioration around $t = 130$ during the rollout. Only $U\text{-}Net_{m8}$ and $U\text{-}Net_{m16}$ are fully stable across the entire rollout of $T = 240$ steps. On Iso, $U\text{-}Net_{m8}$ achieves comparable stability to *ACDM*, with an almost constant rate of change for the entire rollout. Models with shorter rollouts, i.e., *U-Net* and $U\text{-}Net_{m4}$ deteriorate after an initial phase, and longer rollouts prevent effective training for $U\text{-}Net_{m16}$ as explained above. The variants with

Table 6. Accuracy ablation for different training noise standard deviations n.

		$\texttt{Tra}_{\texttt{ext}}$		$\texttt{Tra}_{\texttt{int}}$		$\texttt{Iso}$	
Method	n	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-2})	LSiM (10^{-1})
U-Net	—	3.1 ± 2.1	3.9 ± 2.8	2.3 ± 2.0	3.3 ± 2.8	25.8 ± 35	11.3 ± 3.9
U-Net	1e–4	2.7 ± 1.8	3.9 ± 2.1	1.9 ± 0.8	2.4 ± 2.1	16.0 ± 22	9.6 ± 3.0
U-Net	1e–3	5.6 ± 2.2	3.3 ± 2.5	3.5 ± 1.6	3.0 ± 2.2	36.4 ± 39	12.9 ± 2.2
U-Net	1e–2	1.4 ± 0.8	1.1 ± 0.3	1.8 ± 1.1	1.0 ± 0.4	3.1 ± 0.9	4.5 ± 2.5
U-Net	1e–1	1.8 ± 0.8	1.2 ± 0.2	2.2 ± 2.0	1.2 ± 0.6	3.2 ± 0.5	2.9 ± 0.6
U-Net	1e0	4.0 ± 1.5	1.8 ± 0.3	11.4 ± 6.3	2.9 ± 1.3	16.2 ± 7.8	7.5 ± 2.7
$ACDM_{ncn}$	—	4.1 ± 1.9	1.9 ± 0.6	2.8 ± 1.3	1.7 ± 0.4	18.3 ± 2.5	8.9 ± 1.5
$ACDM_{ncn}$	1e–4	3.8 ± 1.5	2.0 ± 0.3	4.3 ± 2.3	1.7 ± 0.4	14.2 ± 1.7	8.1 ± 1.2
$ACDM_{ncn}$	1e–3	3.6 ± 1.4	2.2 ± 0.3	3.9 ± 2.3	1.8 ± 0.4	11.1 ± 3.8	8.5 ± 1.6
$ACDM_{ncn}$	1e–2	3.6 ± 1.6	1.7 ± 0.4	2.6 ± 2.3	1.3 ± 0.5	26.7 ± 25	12.2 ± 2.8
$ACDM_{ncn}$	1e–1	3.6 ± 1.9	1.5 ± 0.4	2.5 ± 2.2	1.2 ± 0.6	2.8 ± 0.6	4.0 ± 2.2
$ACDM_{ncn}$	1e0	4.2 ± 1.7	1.8 ± 0.4	6.2 ± 2.8	2.0 ± 0.6	11.1 ± 1.4	6.2 ± 0.9

additional pre-training are also included: $U\text{-}Net_{m4,Pre}$ does not substantially improve upon $U\text{-}Net_{m4}$, and $U\text{-}Net_{m8,Pre}$ performs very well, similar to $U\text{-}Net_{m8}$. Only for the longer rollouts in the $U\text{-}Net_{m16,Pre}$ model pre-training clearly helps, as $U\text{-}Net_{m16,Pre}$ is also fully stable.

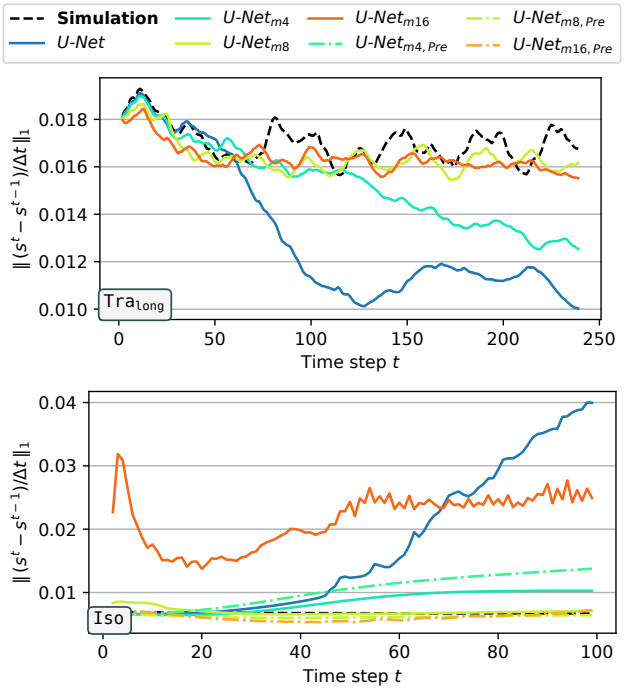

Figure 20. Temporal stability evaluation via error to previous time step for different training rollout lengths m on $\texttt{Tra}_{\texttt{long}}$ (top) and $\texttt{Iso}$ (bottom). Standard deviations are omitted for visual clarity.

Summary Compared to *ACDM*, the variants of *U-Net* with longer training rollouts can achieve similar or slightly higher accuracy and a equivalent temporal stability at a faster inference speed. However, this method requires additional computational resources during training, both in terms of memory over the rollout as well as training time. For example, $U\text{-}Net_{m16,Pre}$ on $\texttt{Iso}$ increases the required training time ($90h$ of pre-training + $260h$ of refinement) by a factor of more than $5.6\times$ at equal epochs and memory compared to $ACDM_{R100}$ as shown in Tab. 3. Naturally, longer training rollouts do not provide *U-Net* with the ability for posterior sampling.

I.3. Ablation on Training Noise

We investigate the usage of training noise (Sanchez-Gonzalez et al., 2020) to stabilize predictions, as it features interesting connections to our method. Instead of generating predictions from noise to achieve temporal stability, this method relies on the addition of noise to the training inputs, to simulate error accumulation during training. In this way, the model adapts to disturbances during training, such that the data shift is reduced once errors inevitably accumulate during the inference rollout, leading to increased temporal stability. We test this approach on *U-Net* and on $ACDM_{ncn}$. The latter evaluation serves as an example to understand if the lost tolerance for error accumulation in $ACDM_{ncn}$, the setup without conditioning noise, can be replaced with training noise. This $ACDM_{ncn}$ version is not intended as a practical architecture as it inherits the drawbacks of both methods, the inference cost from diffusion models, and the overhead and additional hyperparameters from added training noise. For this ablation, we use the $ACDM_{ncn}$ model as described in App. B.1, but add training noise to every model input in the same way as for the *U-Net* with training noise (see App. B.2). Here, *U-Net* or $ACDM_{ncn}$ models trained using training noise with a standard deviation of e.g., $n = 10^{-1}$, are denoted by $U\text{-}Net_{n1e\text{-}1}$ or $ACDM_{ncn,n1e\text{-}1}$ respectively. Prediction examples can be found in Figs. 40 and 41 in App. K.

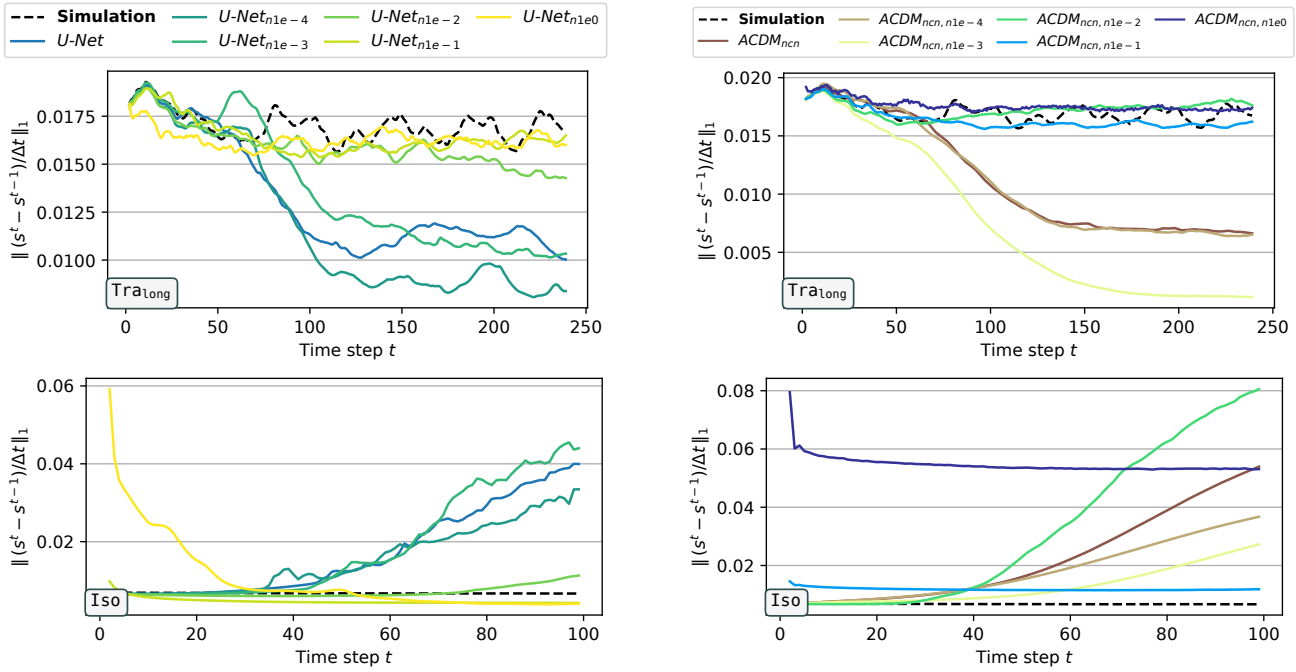

Figure 21. Temporal stability evaluation for different training noise standard deviations n of *U-Net* (left) and *ACDM$_{ncn}$* (right) on `Tra`$_{long}$ (top) and `Iso` (bottom). Standard deviations are omitted for visual clarity.

Accuracy The accuracy of *U-Net* and *ACDM$_{ncn}$* setups with training noise using different standard deviations n is analyzed in Tab. 6. On `Tra`, the accuracy trend is not fully consistent. Small values of n such as 10^{-4} and 10^{-3} occasionally even reduce the final performance, but training noise with a well-tuned standard deviation between 10^{-2} and 10^{-1} does increase accuracy. Choosing very large standard deviations corrupts the training data too much, and reduces accuracy again as expected. The results on the isotropic turbulence experiment show a similar behavior for *U-Net* as well as *ACDM$_{ncn}$*.

Temporal Stability In Fig. 21, we evaluate the temporal stability of models with training noise via the magnitude of the rate of change of s, as detailed in the main paper. On `Tra`$_{long}$ both architectures *U-Net* and *ACDM$_{ncn}$* behave similarly: while training noise with a standard deviation n that is too low does not improve the stability and occasionally even deteriorates it, finding a suitable magnitude is key for stable inference rollouts. In both cases, values of n between 10^{-2} and 10^{-1} produce the best results. Increasing the noise further has detrimental effects, as for example slight overshooting and high-frequency fluctuations occur for *ACDM$_{ncn,n1e0}$* or predictions can diverge early from the simulation for *U-Net$_{n1e0}$*. On `Iso`, a similar stabilizing effect from training noise can be observed, given the noise magnitude is tuned sufficiently: While lower standard deviations barely alter the time point $t = 40$, where predictions diverge from the reference simulation, too much training noise already causes major problems at the very beginning of the prediction. This behavior can also be observed on a spatial spectral analysis via the TKE in Fig. 22, where

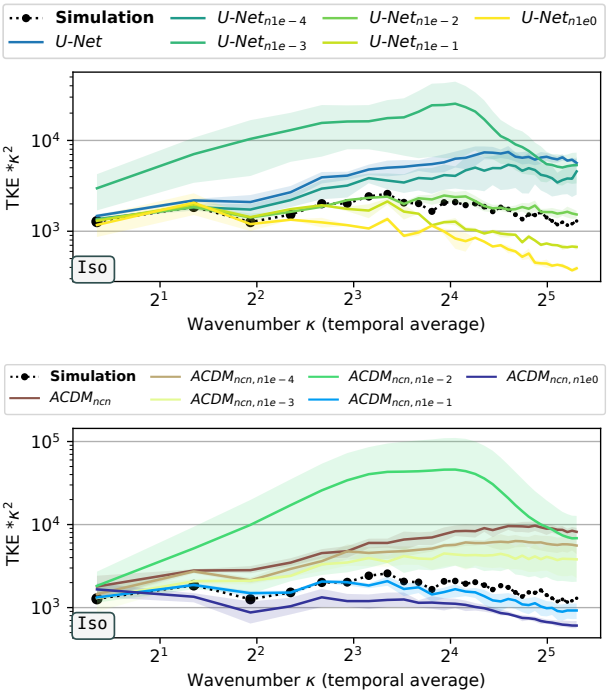

Figure 22. Spatial frequency analysis via the turbulent kinetic energy (TKE) on a sequence from `Iso` with $z = 300$ for the training noise ablations on *U-Net* (top) and *ACDM$_{ncn}$* (bottom).

Table 7. Accuracy ablation for training with LSiM losses of different strengths λ.

		$\texttt{Tra}_{\texttt{ext}}$		$\texttt{Tra}_{\texttt{int}}$		$\texttt{Iso}$	
Method	λ	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-2})	LSiM (10^{-1})
U-Net	—	3.1 ± 2.1	3.9 ± 2.8	2.3 ± 2.0	3.3 ± 2.8	25.8 ± 35	11.3 ± 3.9
U-Net	1e–5	4.2 ± 2.9	4.5 ± 3.0	2.6 ± 2.2	2.1 ± 2.0	67.4 ± 75.7	12.4 ± 3.8
U-Net	1e–4	2.3 ± 1.2	3.7 ± 2.6	1.6 ± 1.4	2.0 ± 1.8	12.3 ± 9.3	11.8 ± 2.5
U-Net	1e–3	2.9 ± 1.9	1.7 ± 0.8	2.2 ± 2.3	1.5 ± 0.9	6.3 ± 3.1	9.4 ± 2.8
U-Net	1e–2	4.5 ± 1.3	3.5 ± 1.1	3.0 ± 2.3	1.8 ± 0.9	$0.1b \pm 0.2b$	15.3 ± 1.2
U-Net	1e–1	5.8 ± 1.8	3.0 ± 0.8	5.2 ± 1.9	2.3 ± 0.6	$12b \pm 29b$	15.0 ± 1.0
U-Net	1e0	6.8 ± 1.5	4.8 ± 1.1	6.6 ± 3.0	2.4 ± 0.7	$17b \pm 552b$	14.9 ± 1.0

the training noise can balance predictions between under- and overshooting. For both *U-Net* and *ACDM*$_{ncn}$, training noise with a suitable magnitude can result in a comparable temporal stability to *ACDM*, that includes noise on the conditioning.

Summary Training *U-Net* with training noise can achieve similar or slightly higher accuracy and a competitive temporal stability compared to *ACDM*. While this method exhibits faster inference speeds, it does rely on the additional noise variance hyperparameter, that can even reduce performance if not tuned well. Furthermore, training noise does not provide deterministic models with the ability for posterior sampling. Interestingly, the lost error tolerance of the *ACDM*$_{ncn}$ architecture without conditioning noise, can be mostly restored with training noise of suitable magnitude.

I.4. Ablation on Training with an LSiM Loss

In this section, we investigate usage of the LSiM metric (Kohl et al., 2020) as an additional loss term, similar to perceptual losses in the computer vision domain (Dosovitskiy & Brox, 2016; Johnson et al., 2016). This means, in addition to training *U-Net* with an MSE loss as above, the differentiable learned LSiM metric model is also used during back-propagation. Given a predicted state s^t and the corresponding ground truth state $\hat{s}^t$, we evaluate the training loss as

$$\mathcal{L}_{MSE+LSiM} = \left(s^t - \hat{s}^t\right)^2 + \lambda * \text{LSiM}(s^t, \hat{s}^t)$$

while leaving the inference of the models untouched. To use LSiM, each field from both states is individually normalized to $[0, 255]$. The resulting loss values are aggregated with an average operation across fields. Fields containing the scalar simulation parameters are not evaluated with this metric. In the following, the impact of λ, the weight that controls the influence of the LSiM loss, is investigated. *U-Net* models trained with e.g., $\lambda = 10^{-1}$, are denoted by *U-Net*$_{\lambda 1e\text{-}1}$.

Accuracy In terms of accuracy, adding very small amounts of the LSiM term with $\lambda = 10^{-5}$ to the MSE loss does decrease performance, most likely due to suboptimal gradient signals through the additional steps during back-propagation, as shown in Tab. 7. Similarly, adding too much, such that it predominantly influences the overall loss causes problems. Especially on $\texttt{Iso}$, this causes models to aggressively diverge after $30 - 40$ prediction steps, leading to errors in the range of 10^9 (b) in Tab. 7. As expected, choosing a suitable loss magnitude around $\lambda = 10^{-3}$ substantially reduces errors in terms of LSiM across test sets. However, the added loss term does also improve performance in terms of MSE, as similarly observed in the image domain (Dosovitskiy & Brox, 2016; Johnson et al., 2016).

Temporal Stability In line with the accuracy results, *U-Net*$_{\lambda 1e\text{-}3}$ exhibits improved temporal stability compared to

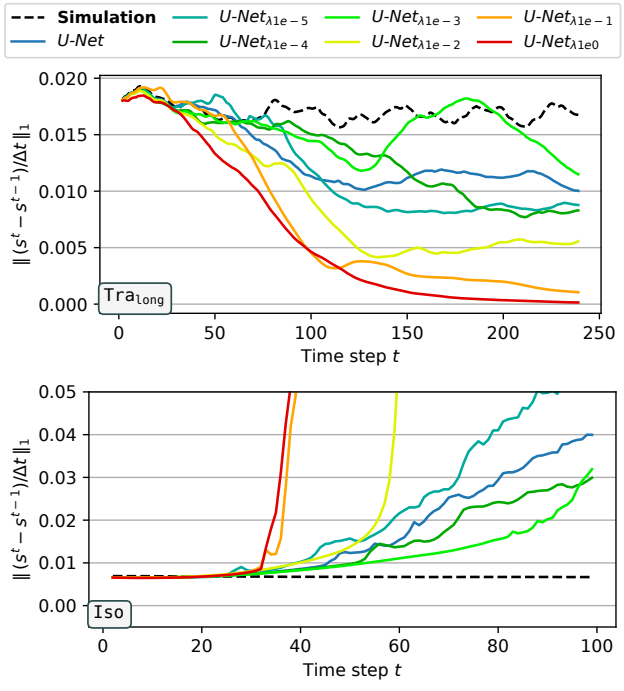

Figure 23. Temporal stability evaluation of *U-Net* for training with LSiM losses of different strengths λ on $\texttt{Tra}_{\texttt{long}}$ (top) and $\texttt{Iso}$ (bottom). Standard deviations are omitted for visual clarity.

Table 8. Accuracy of the "modern" U-Net architecture compared to *DFP*.

	$\texttt{Tra}_{\text{ext}}$		$\texttt{Tra}_{\text{int}}$		$\texttt{Iso}$	
Method	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-2})	LSiM (10^{-1})
U-Net	3.1 ± 2.1	3.9 ± 2.8	2.3 ± 2.0	3.3 ± 2.8	25.8 ± 35	11.3 ± 3.9
DFP	4.5 ± 1.3	3.9 ± 0.7	4.8 ± 2.1	3.6 ± 1.7	5.1 ± 1.3	5.1 ± 2.0
ACDM	2.3 ± 1.4	1.3 ± 0.3	2.7 ± 2.1	1.3 ± 0.6	3.7 ± 0.8	3.3 ± 0.7
DFP_{ACDM}	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*	*NaN*

U-Net as displayed in Fig. 23. Choosing unsuitable λ causes models to diverge earlier from the reference trajectory when evaluating the difference between predictions steps, for both $\texttt{Tra}_{\text{long}}$ and $\texttt{Iso}$. When analyzing the frequency behavior of the models trained with LSiM in Fig. 24 the results are similar: Improved performance across the frequency band can be observed for $U\text{-}Net_{\lambda 1e\text{-}3}$, while smaller values of λ are less potent and can be detrimental or only slightly beneficial compared to *U-Net*.

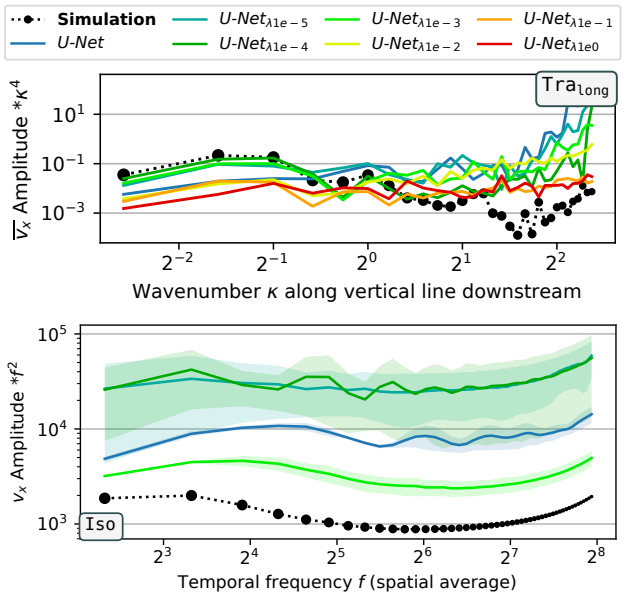

Figure 24. Spatial frequency along a vertical line downstream on $\texttt{Tra}_{\text{long}}$ (top) and temporal frequency analysis on a sequence from $\texttt{Iso}$ with $z = 300$ (bottom) for the LSiM loss ablation models.

Summary Training *U-Net* with LSiM as an additional loss term, can increase accuracy, temporal stability, and frequency behavior across evaluations. However, the resulting models are neither competitive compared to other stabilization techniques discussed above, such as training rollouts or training noise, nor to the proposed diffusion architecture.

I.5. Ablation on U-Net Modernizations

As described in App. B.2, our U-Net implementation follows established diffusion model architectures, that contain a range of modernizations compared to the original approach proposed by Ronneberger et al. (2015). Here, we compare to a more traditional U-Net architecture which is known to work well for fluid problems. We adapted the DFP model implementation[7] of Thuerey et al. (2020) for our settings. The architecture features:

- batch normalization instead group normalization, and no attention layers in the blocks,
- six downsampling blocks consisting of strided convolutions and leaky ReLU layers,
- six upsampling blocks consisting of convolution, bilinear upsamling, and ReLU layers,
- six feature map levels with spatial sizes of 64×32, 32×16, 16×8, 8×4, 4×2, and 2×1,
- an increasing number of channels for deeper features, i.e., 72, 72, 144, 288, 288, and 288.

It is trained as a direct one-step predictor (*DFP*) in the same way as described in App. B.2, as well as employing it in the diffusion setup as a backbone architecture (DFP_{ACDM}). In both cases, we keep all other hyperparameters identical with the corresponding baseline architecture.

Accuracy Table 8 shows a comparison of both architectures compared to *U-Net* and *ACDM* on our more challenging data sets $\texttt{Tra}$ and $\texttt{Iso}$. Training DFP_{ACDM} as a diffusion backbone (with additional time embeddings for the diffusion step r as discussed in App. B.1) failed to generalize beyond the first few prediction time steps across test sets in our experiments. This highlights the general usefulness of the recently introduced modernizations to the U-Net architecture. There is a noticeable drop in accuracy on $\texttt{Tra}$ for *DFP* compared to *U-Net*, but it performs clearly better than *U-Net* on $\texttt{Iso}$, however still lacking compared to *ACDM*. This unexpected trend in accuracy is mainly caused by the different rollout behavior of these architectures discussed in the following.

Temporal Stability Figure 25 shows temporal stability evaluations of the variants on $\texttt{Tra}_{\text{long}}$ and $\texttt{Iso}$, that illustrate the different rollout behavior of *DFP* compared to

[7] https://github.com/thunil/Deep-Flow-Prediction

U-Net depending on the data set. On $\texttt{Tra}_{\texttt{long}}$ on the top, *DFP* diverges earlier and more substantially compared to *U-Net*, when measured via the difference to the previously predicted time step. However, the rollout behavior is different on `Iso`, as illustrated via the Pearson correlation coefficient to the ground truth trajectory on the bottom in Fig. 25. The simpler *DFP* model decorrelates more quickly for the first 50 steps, while keeping a relatively constant decorrelation rate. *U-Net* is initially more in line with the reference, however it sharply decreases after about 50 steps, meaning errors accumulate more quickly after an initial phase of higher stability.

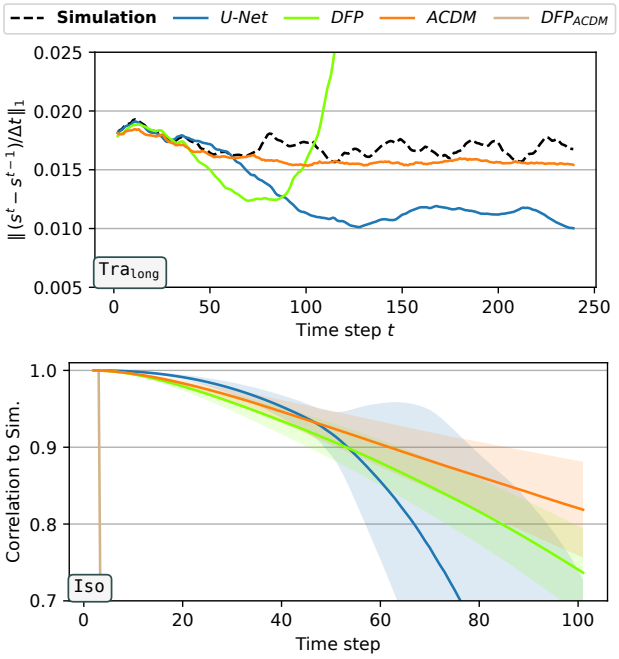

Figure 25. Temporal stability evaluation via error to previous time step on $\texttt{Tra}_{\texttt{long}}$ (top) and the correlation to the ground truth on `Iso` (bottom) for the U-Net modernization ablation.

Summary We found the recently proposed architecture modernizations to U-Nets to be an important factor, when employing them as backbones in a diffusion-based setup. For some direct prediction cases, the modernizations can delay diverging behavior due to unrolling during inference to some degree. On other data, using no modernizations can be beneficial for longer rollouts in direct prediction setting, but this comes at the costs of less initial accuracy, and lacking capacities as a diffusion backbone.

I.6. Ablations on PDE-Refiner

Here, we investigate PDE-Refiner in more detail, especially with respect to the number of refinement steps R and the minimum noise variance σ, its key hyperparameters. We sweep over combinations of $R \in \{2, 4, 8\}$ and $\sigma \in \{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$, and report accuracy, temporal stability, and posterior sampling results. Due to computational constraints for this large sweep, only one model per combination is trained. Five samples from each model are considered, as above. We denote models trained with e.g., $R = 2$ and $\sigma = 10^{-3}$ by $\textit{Refiner}_{R2,\sigma 1e\text{-}3}$ in the following. Prediction examples can be found in Figs. 42 and 43 in App. K.

Accuracy Table 9 evaluates the accuracy of these PDE-Refiner variants compared to *ACDM* and *U-Net* on our data sets `Tra` and `Iso`. Overall, the performance of *Refiner* across data sets, number of refinement steps R, and noise variances σ is highly unpredictable. There is neither a clear accuracy trend over few or many refinement steps, nor high or low noise variance. Furthermore, a high accuracy on `Tra` is not directly correlated with a high accuracy on `Iso` either. As *Refiner* essentially improves upon one-step predictions of *U-Net* via additional refinement steps, the results of a direct comparison are interesting: On `Tra`, while *Refiner* consistently outperforms *U-Net* in terms of LSiM, it just as consistently remains worse in terms of the MSE across hyperparameter combinations. We hypothesize that these results are linked to the fundamentally different spectral behavior of *Refiner* described by Lippe et al. (2023), but further research is required in this direction. On `Iso`, *Refiner* either improves upon *U-Net* or substantially diverges (marked in grey in Tab. 9), especially for small σ. Overall, PDE-Refiner is less effective than the stabilization techniques discussed in Appendices I.2 and I.3 in terms of accuracy improvements, and thus consistently falls short with respect to *ACDM* across test sets and hyperparameter combinations.

Temporal Stability To investigate the temporal stability of *Refiner*, we analyze the difference to the previous time step in Fig. 26. First, it is shown that there are combinations of R and σ that substantially improve the rollout stability of *Refiner* compared to *U-Net*, confirming the results from Lippe et al. (2023). However, as observed in terms of accuracy above, there is no consistent trend across hyperparameters and data sets. Especially, finding a suitable minimum noise variance σ depends on both, data set and number of refinement steps R: While $\sigma = 10^{-6}$ works best on $\texttt{Tra}_{\texttt{long}}$ for $R = 2$, $\sigma = 10^{-7}$ is ideal for $R = 8$. On `Iso`, $R = 2$ only works with $\sigma = 10^{-3}$, $R = 4$ requires $\sigma = 10^{-4}$, and $R = 8$ is most stable with $\sigma = 10^{-5}$. This unpredictable behavior with respect to important hyperparameters makes PDE-Refiner resource-intensive and difficult to employ in practice. The best *Refiner* variants on `Iso`, while more stable compared to *U-Net*, are nevertheless showing signs of instabilities around $t = 70$. This means the refinement increases stability, but still falls short compared to the other discussed stabilization techniques.

Posterior Sampling As PDE-Refiner relies on deterministic predictions combined with probabilistic refinements,

Table 9. Accuracy comparison for PDE-Refiner using different refinement steps R and noise variances σ.

			$\texttt{Tra}_{\texttt{ext}}$		$\texttt{Tra}_{\texttt{int}}$		$\texttt{Iso}$	
Method	R	σ	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-2})	LSiM (10^{-1})
ACDM	—	—	2.3 ± 1.4	1.3 ± 0.3	2.7 ± 2.1	1.3 ± 0.6	3.7 ± 0.8	3.3 ± 0.7
U-Net	—	—	3.1 ± 2.1	3.9 ± 2.8	2.3 ± 2.0	3.3 ± 2.8	25.8 ± 35	11.3 ± 3.9
Refiner	2	1e–3	3.3 ± 1.3	1.4 ± 0.3	3.9 ± 1.6	1.4 ± 0.3	6.1 ± 1.9	7.2 ± 1.6
Refiner	2	1e–4	12.7 ± 2.9	4.2 ± 0.5	10.1 ± 1.5	2.4 ± 0.3	$0.1m \pm 0.3m$	12.5 ± 5.2
Refiner	2	1e–5	4.8 ± 1.4	2.6 ± 0.3	4.0 ± 3.1	2.1 ± 0.5	$3.3e30$	15.2 ± 0.9
Refiner	2	1e–6	5.0 ± 1.9	2.0 ± 0.3	3.6 ± 2.6	1.9 ± 0.4	$0.1m \pm 0.2m$	16.1 ± 1.0
Refiner	2	1e–7	13.6 ± 9.9	6.1 ± 4.0	54.6 ± 68.7	6.7 ± 5.0	$22k \pm 13k$	14.9 ± 0.9
Refiner	4	1e–3	5.3 ± 0.8	3.2 ± 0.4	6.0 ± 1.2	2.6 ± 0.4	5.1 ± 1.8	4.7 ± 0.8
Refiner	4	1e–4	3.4 ± 2.0	1.9 ± 0.3	5.7 ± 2.4	1.9 ± 0.5	7.0 ± 3.1	5.0 ± 1.0
Refiner	4	1e–5	7.0 ± 1.7	2.7 ± 0.4	3.1 ± 0.8	1.7 ± 0.2	4.9 ± 2.0	7.6 ± 2.1
Refiner	4	1e–6	3.5 ± 1.1	2.1 ± 0.5	8.8 ± 0.9	4.3 ± 2.1	66.1 ± 38.4	11.7 ± 0.7
Refiner	4	1e–7	5.4 ± 1.0	3.1 ± 0.2	8.3 ± 2.2	2.7 ± 0.2	$1.9e18$	14.8 ± 1.0
Refiner	8	1e–3	7.1 ± 1.5	3.5 ± 0.4	4.4 ± 1.8	2.7 ± 0.4	5.5 ± 1.3	6.9 ± 1.0
Refiner	8	1e–4	13.8 ± 2.3	5.0 ± 0.5	8.6 ± 4.2	2.4 ± 0.7	5.1 ± 1.3	5.9 ± 1.1
Refiner	8	1e–5	6.3 ± 1.1	3.5 ± 0.4	6.0 ± 1.8	2.4 ± 0.6	4.7 ± 0.7	5.4 ± 1.2
Refiner	8	1e–6	3.1 ± 1.3	2.2 ± 0.2	6.4 ± 2.1	2.0 ± 0.4	$0.1k \pm 0.3k$	6.1 ± 4.3
Refiner	8	1e–7	4.3 ± 1.4	2.1 ± 0.3	3.3 ± 1.2	1.6 ± 0.3	88 ± 70	6.2 ± 1.9

achieving a broad and diverse posterior distribution is difficult. In Fig. 27, we visualize posterior samples for $\texttt{Tra}_{\texttt{long}}$ from *ACDM* and *Refiner* with $R \in \{2, 4, 8\}$ and $\sigma \in \{10^{-5}, 10^{-6}, 10^{-7}\}$. While *ACDM* creates a broad range of samples as discussed in the main paper above, *Refiner*$_{R2,\sigma 1e\text{-}6}$, *Refiner*$_{R4,\sigma 1e\text{-}6}$, and *Refiner*$_{R4,\sigma 1e\text{-}7}$ do not create any noticeable variance. While additional refinement steps slightly improve the spread across samples, even *Refiner*$_{R8,\sigma 1e\text{-}6}$ can only create minor differences with very similar spatial structures. Note that the *Refiner* models are in general unable to create the detailed shockwaves below the cylinder that are found in the simulation and the *ACDM* samples. In addition, unphysical predictions after longer rollouts can be observed across refinement steps and noise variances in the visualizations in Fig. 42. Using very larger values of σ should theoretically allow *Refiner* to focus on a larger range of frequencies. However, this increased the stability issues further and did not substantially improve the quality or diversity of posterior samples over Fig. 27.

Summary While the stability benefits of a well-tuned setup with PDE-Refiner compared to a simple one-step prediction with *U-Net* are highly desirable and can be achieved with less inference overhead compared to *ACDM*, the method has several disadvantages: We found the setup to be very sensitive regarding changes to refinement steps, data set, or noise variance. This means, a large amount of computational resources are required for parameter tuning, which is crucial to obtain good results. Suboptimal combinations of refinement steps and noise variance show substantially degraded performance compared to *U-Net* in our experiments, and even tuned setup did exhibit instabilities across training runs and model samples. Furthermore, *Refiner* is less accuracy, and has limits in terms of the posterior sampling compared to a more direct application of diffusion models in *ACDM*.

J. Prediction Examples

Over the following pages, prediction examples from all analyzed methods in the main paper are displayed. Shown are the different fields contained in an exemplary test sequence from each experiment. Figures 28 and 29 feature the $\texttt{Inc}_{\texttt{var}}$ case, Figs. 30 to 32 contain an example from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$, and Figs. 33 to 35 display a sequence from $\texttt{Iso}$ with $z = 280$. Supplementary videos of model predictions for some example sequences from each data set are provided alongside this work, as they can visualize several aspects like temporal stability, temporal coherence, and visual quality better than still images. We also include videos of posterior samples from the probabilistic architectures, and a temporal coherence analysis of *ACDM*. All videos can be found alongside this work at https://ge.in.tum.de/publications/2023-acdm-kohl/.

K. Ablation Study Prediction Examples

Below the prediction examples for the model architectures, we display prediction examples from different ablation study models provided in Appendices I.1 to I.3 and I.6. Shown are the pressure field from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$, as well as a vorticity sequence from $\texttt{Iso}$ with $z = 280$.

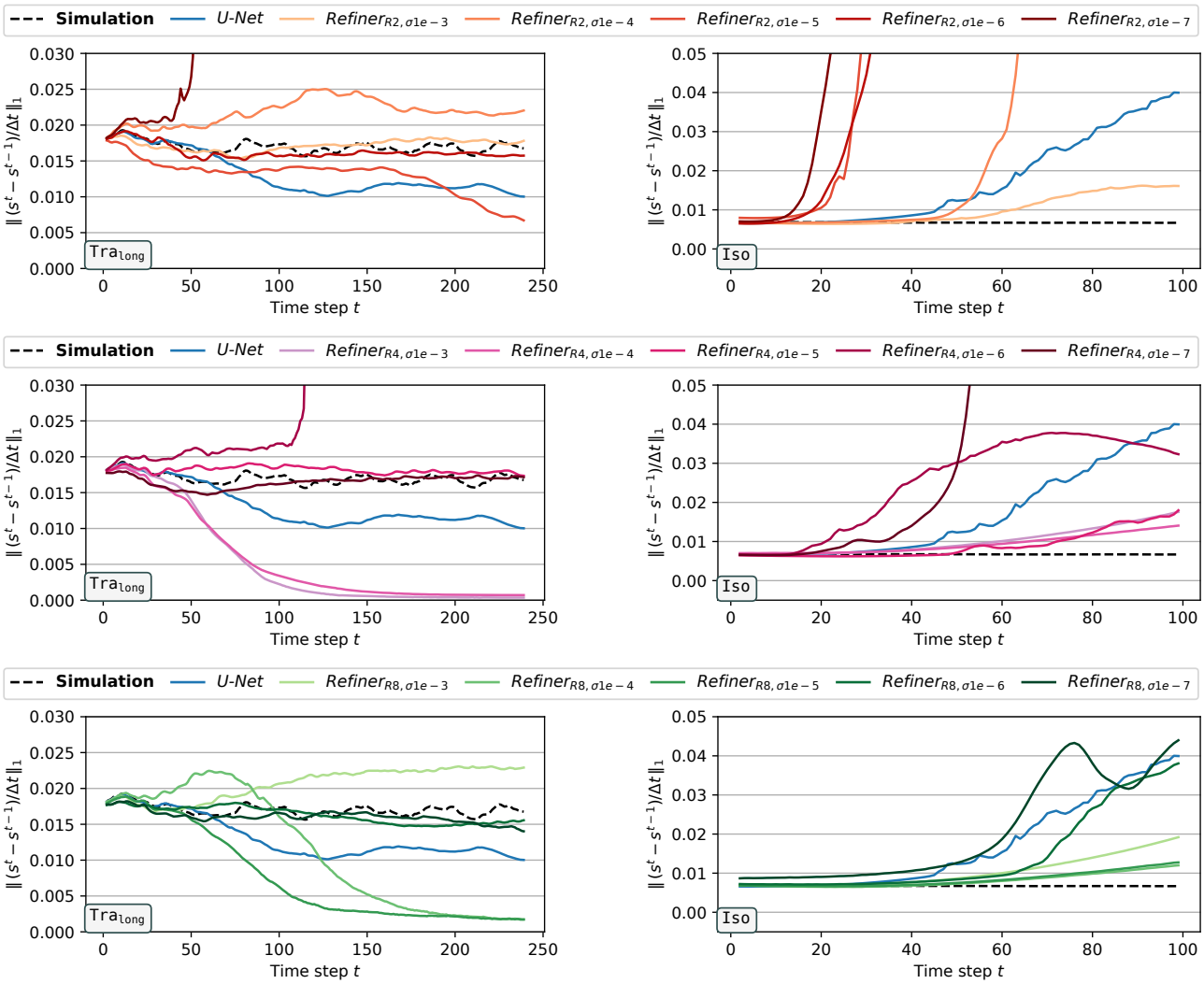

Figure 26. Temporal stability evaluation via error to previous time step on $\texttt{Tra}_{\texttt{long}}$ (left) and on `Iso` (right) for PDE-Refiner with different hyperparameter combinations of refinement steps R and noise variances σ. Standard deviations are omitted for visual clarity. The temporally most stable *Refiner* configuration is highly inconsistent, and for a given R depends on the data set and noise variance.

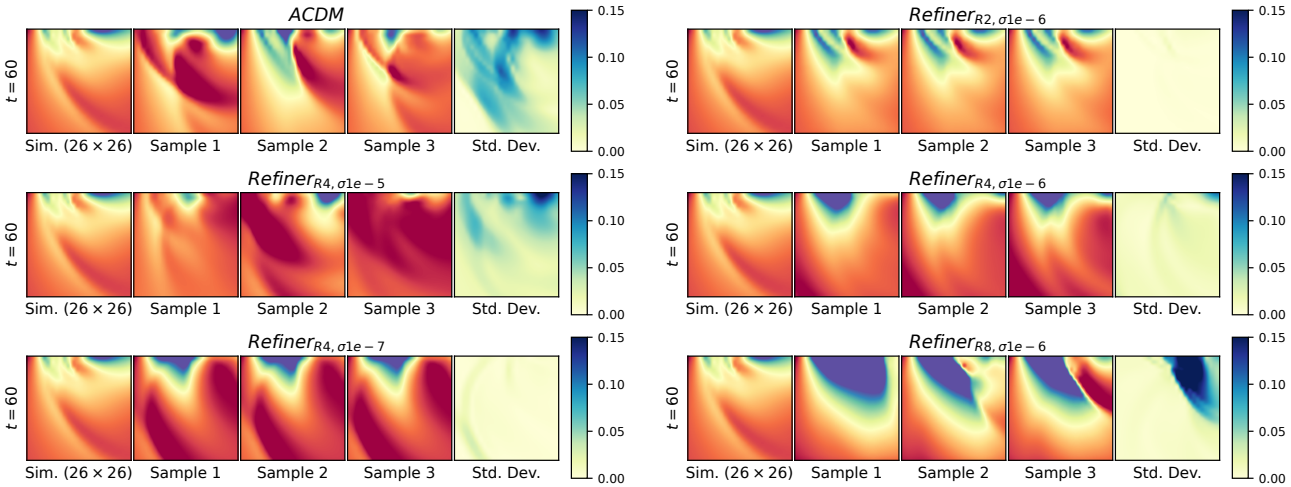

Figure 27. Posterior samples on $\texttt{Tra}_{\texttt{long}}$ from *ACDM* (top left) compared to PDE-Refiner ablation models with different refinement steps R and noise variances σ (other). *Refiner* lacks sample diversity and quality compared to *ACDM* across values for R and σ.

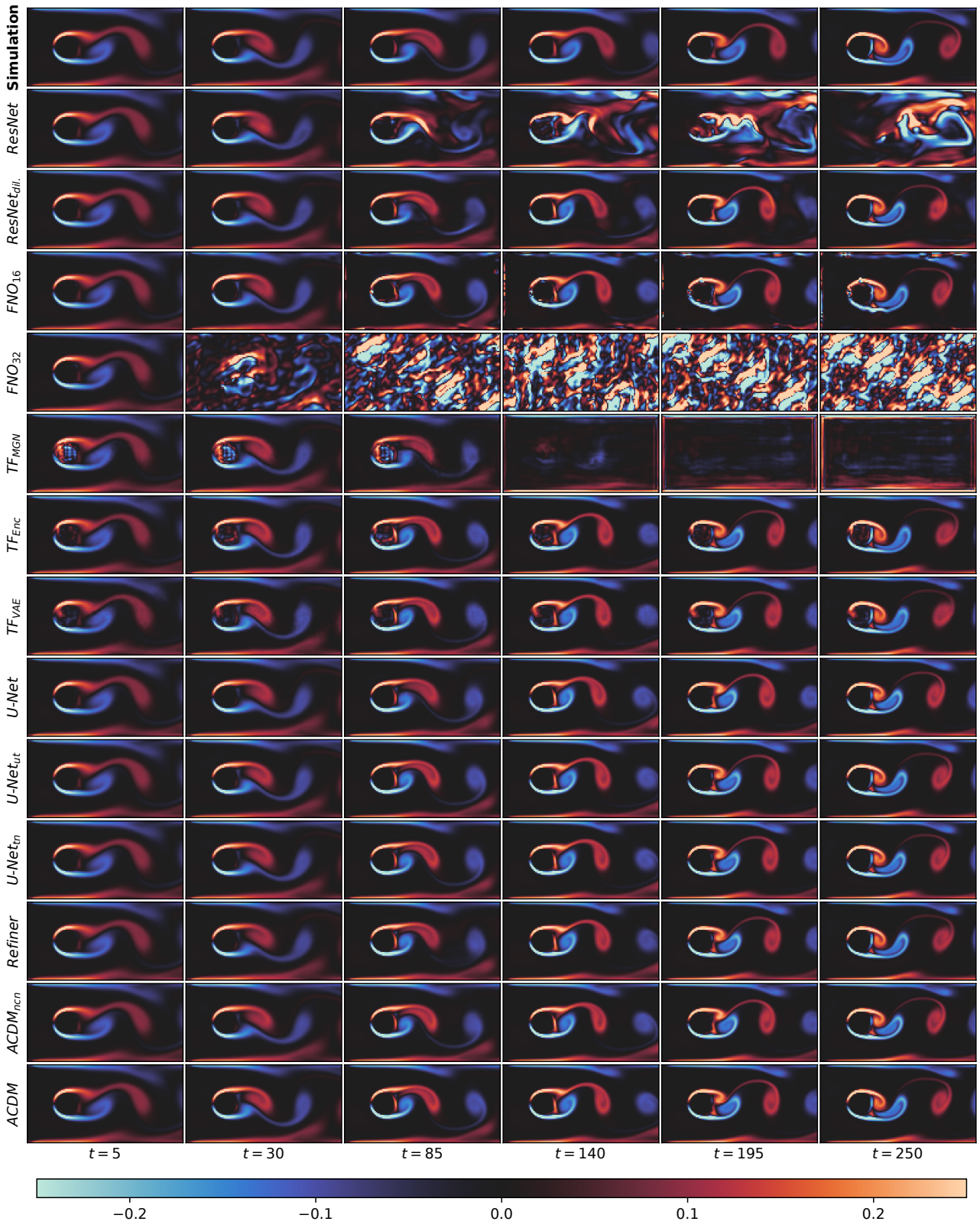

Figure 28. Vorticity predictions for the $\texttt{Inc}_{\texttt{var}}$ sequence.

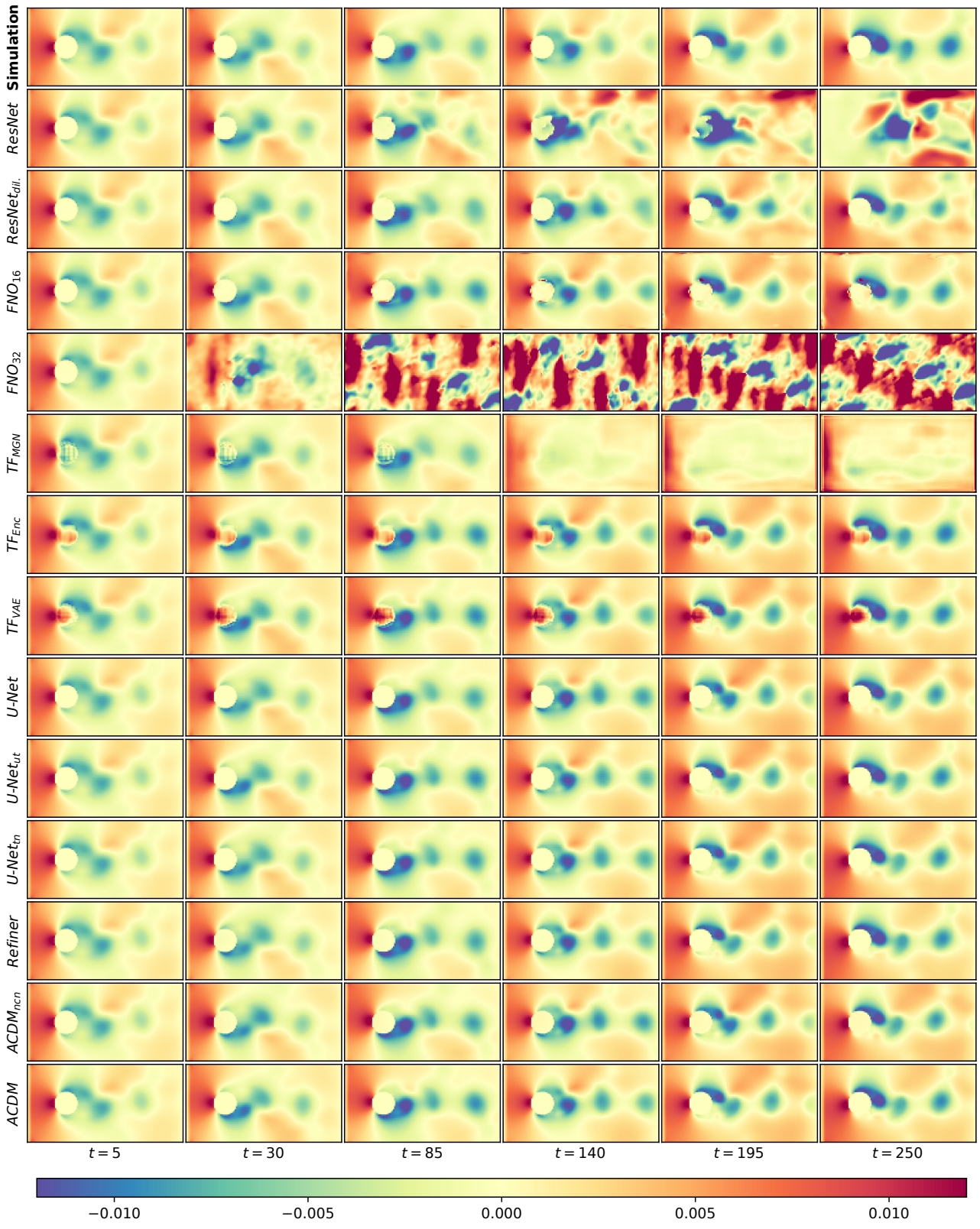

Figure 29. Pressure predictions for the $\texttt{Inc}_{\texttt{var}}$ sequence.

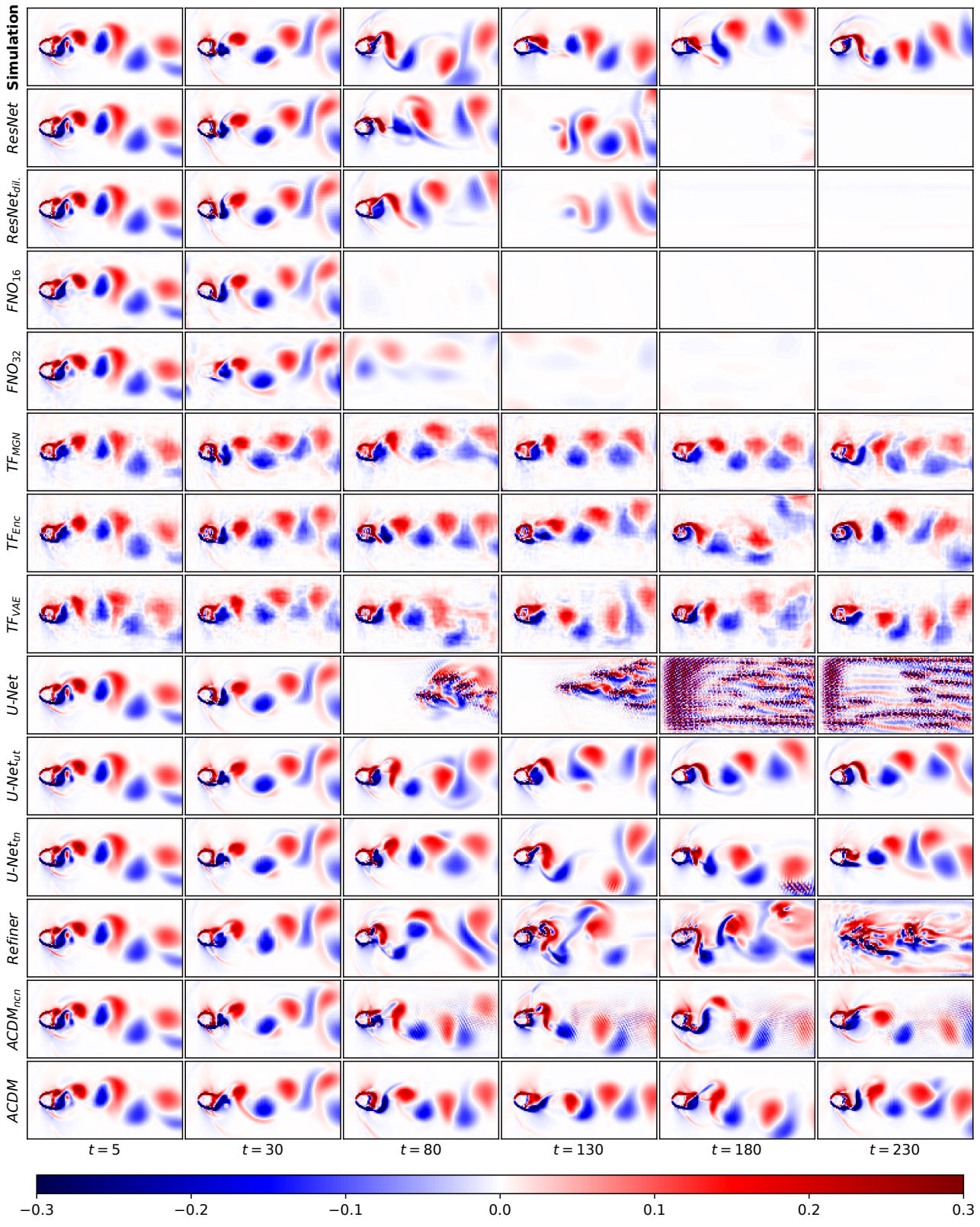

Figure 30. Vorticity predictions for an example sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$.

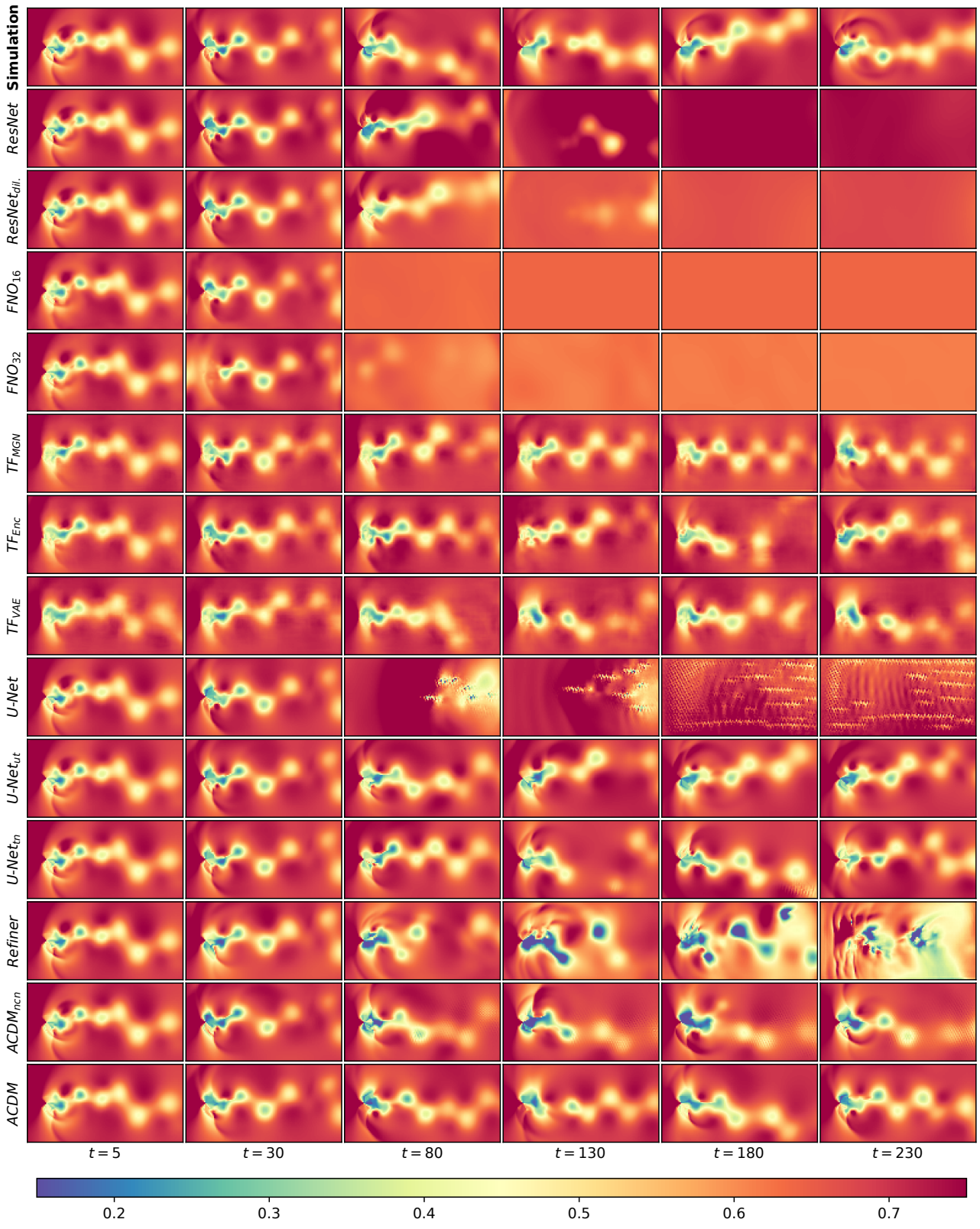

Figure 31. Pressure predictions for an example sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$.

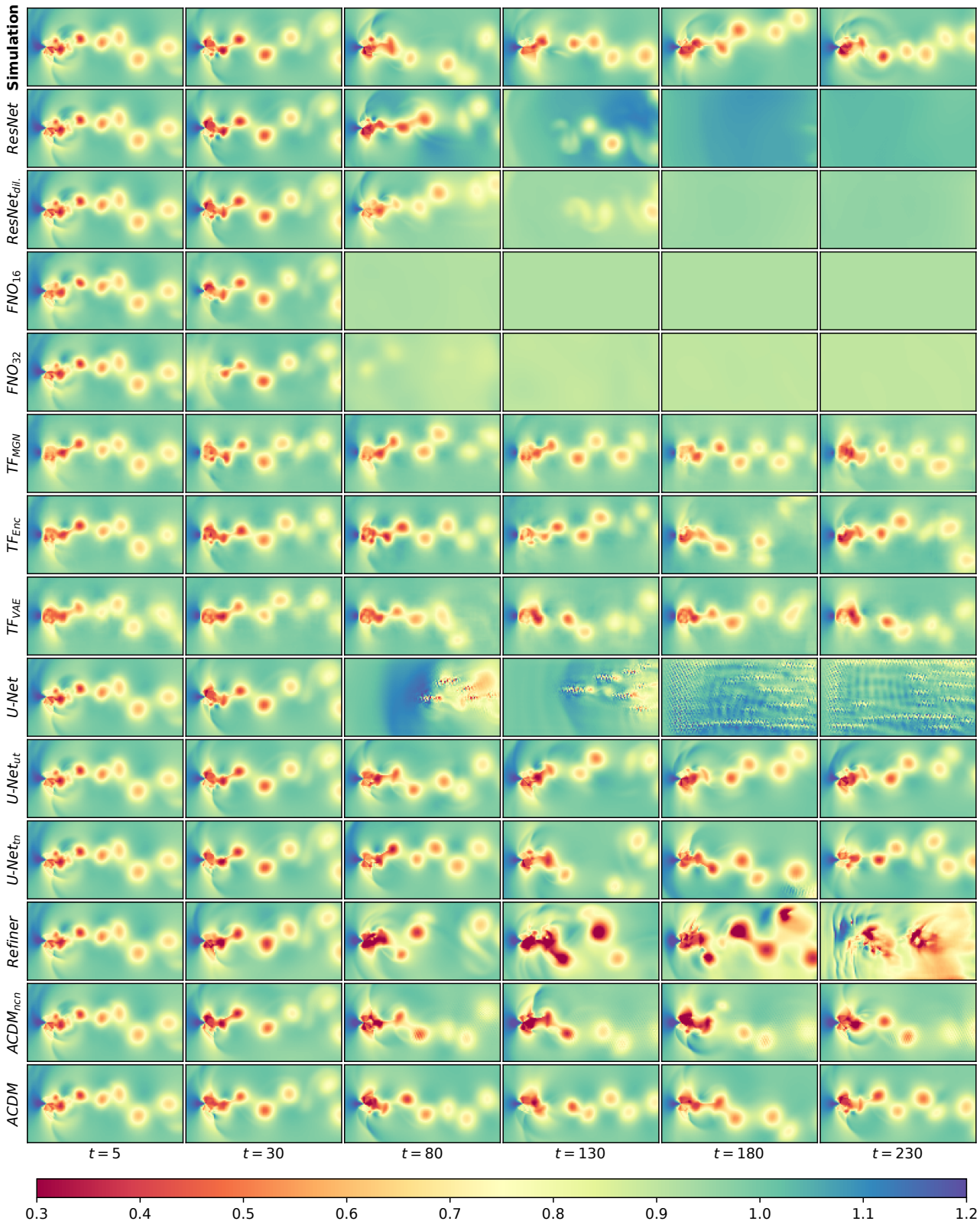

Figure 32. Density predictions for an example sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$.

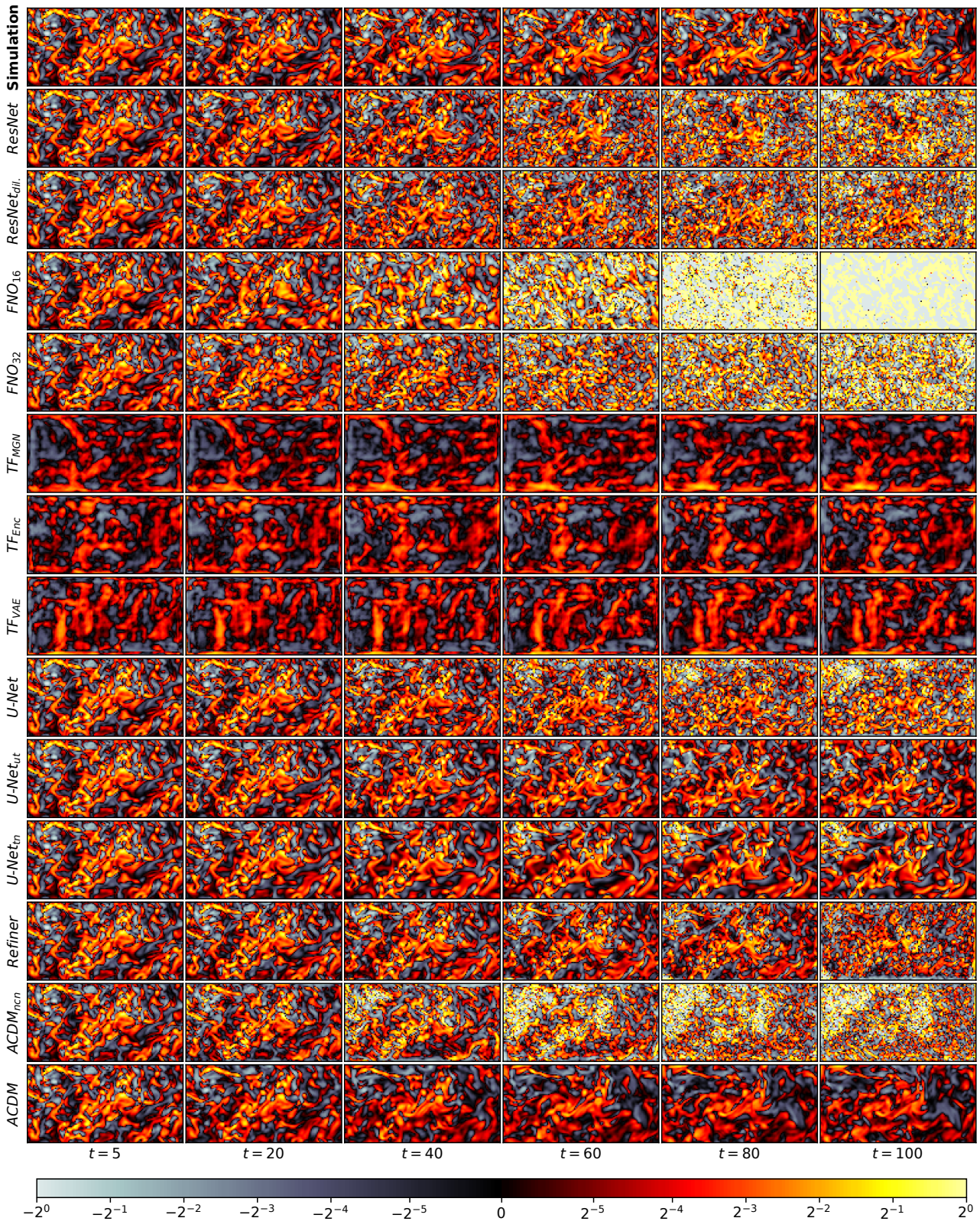

Figure 33. Vorticity predictions (only z-component) for an example sequence from `Iso` with $z = 280$.

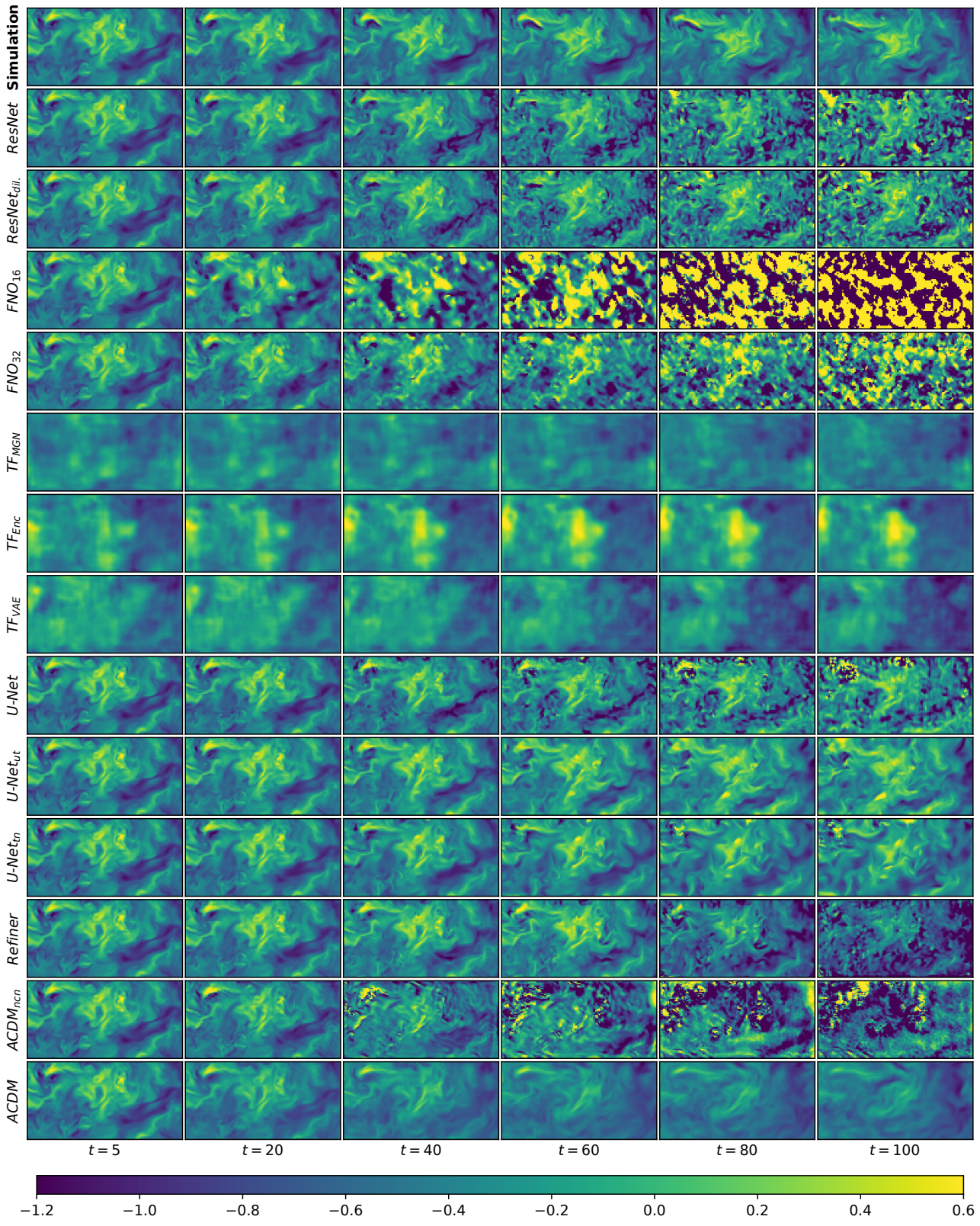

Figure 34. Z-velocity predictions for an example sequence from `Iso` with $z = 280$.

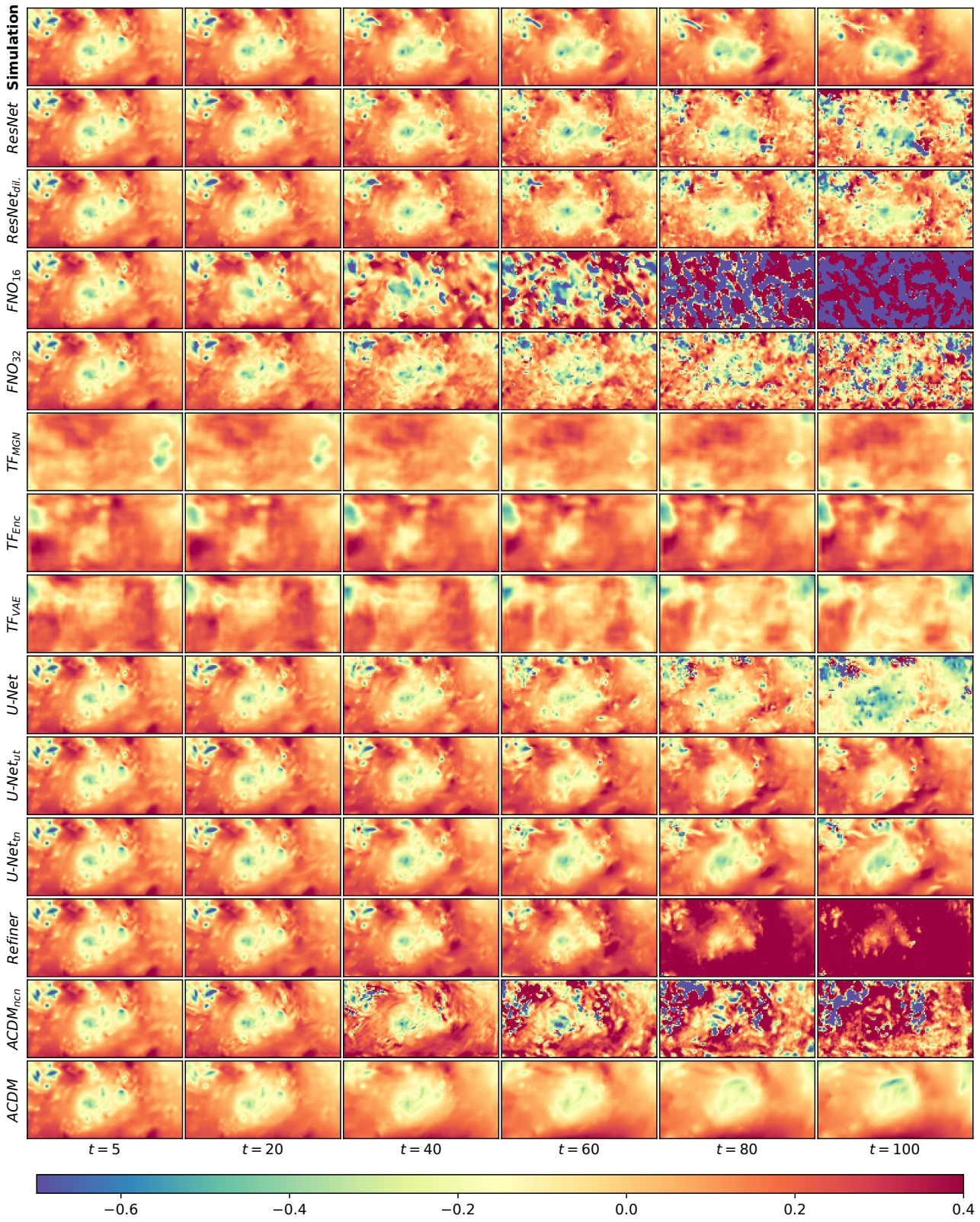

Figure 35. Pressure predictions for an example sequence from `Iso` with $z = 280$.

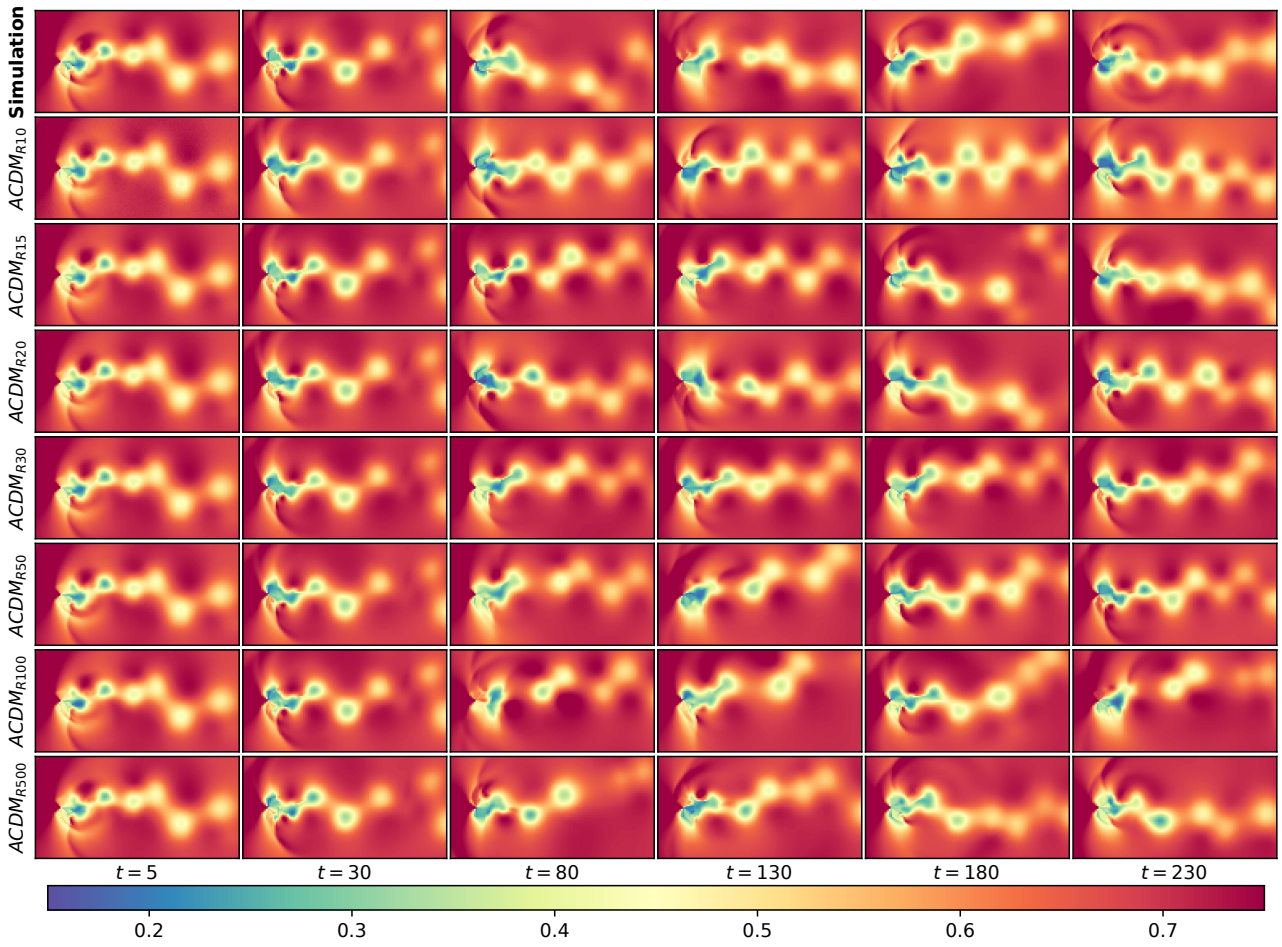

Figure 36. Diffusion Step Ablation (see App. I.1): Pressure predictions from $\texttt{Tra}_{\texttt{long}}$.

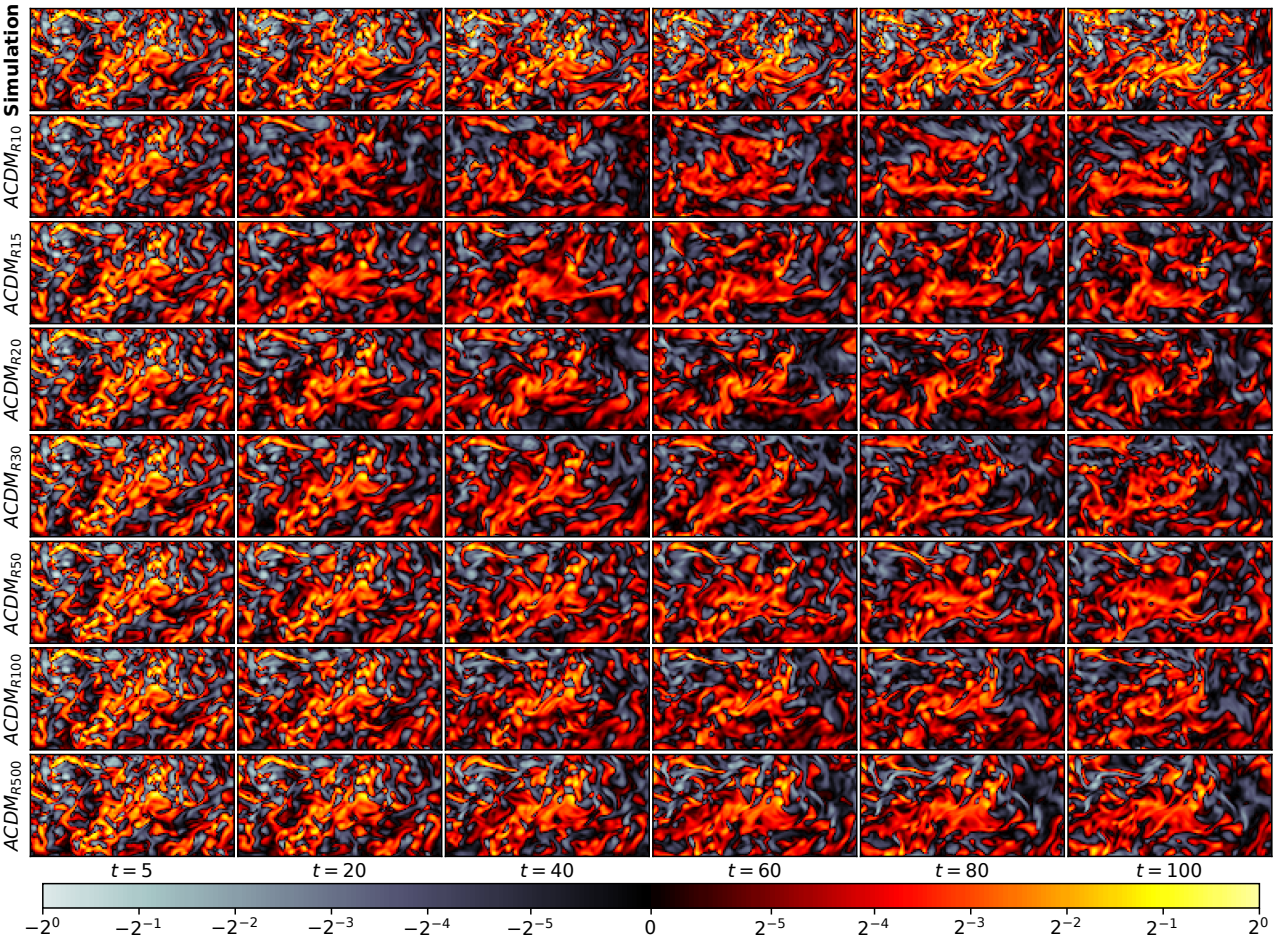

Figure 37. Diffusion Step Ablation (see App. I.1): Vorticity predictions from `Iso`.

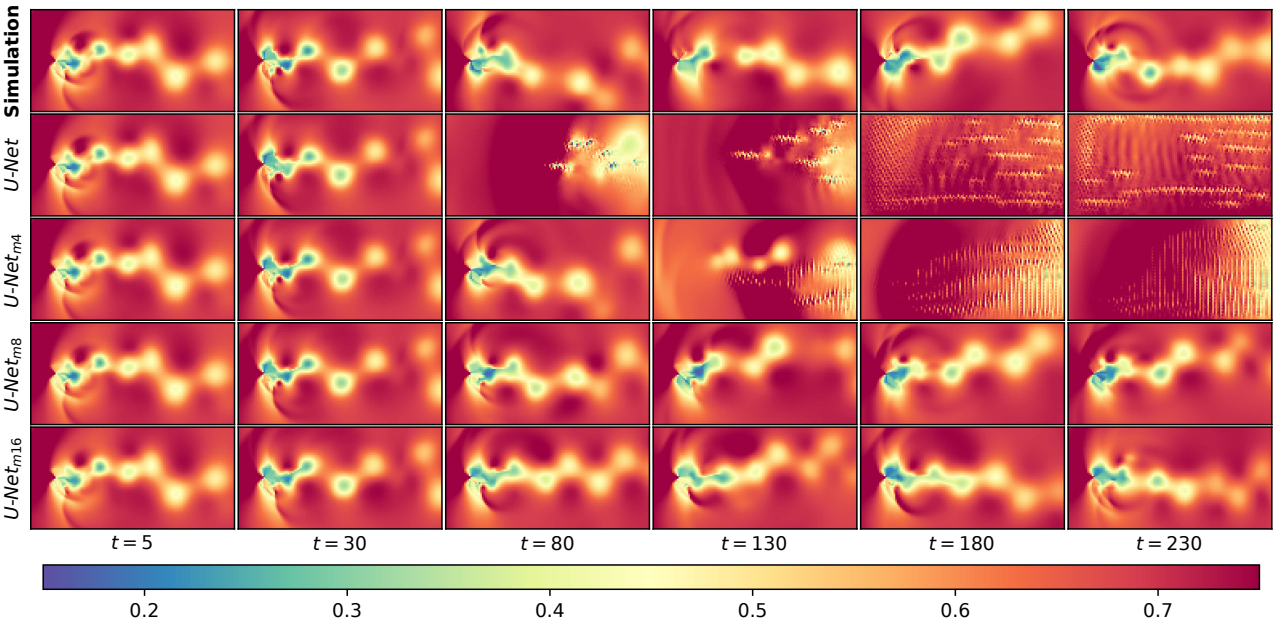

Figure 38. Training Rollout Ablation (see App. I.2): Pressure predictions from $\texttt{Tra}_{\texttt{long}}$.

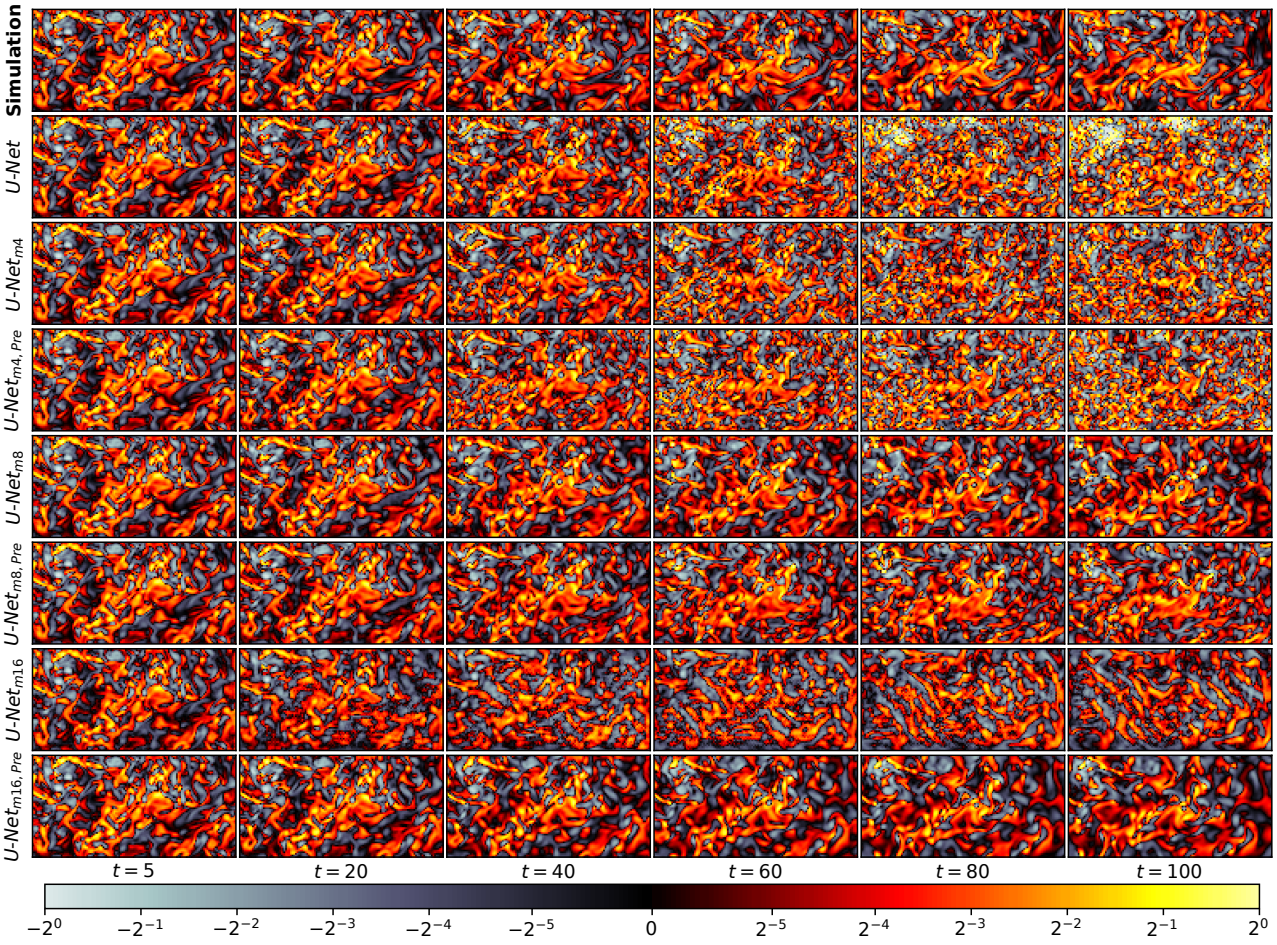

Figure 39. Training Rollout Ablation (see App. I.2): Vorticity predictions from $\texttt{Iso}$.

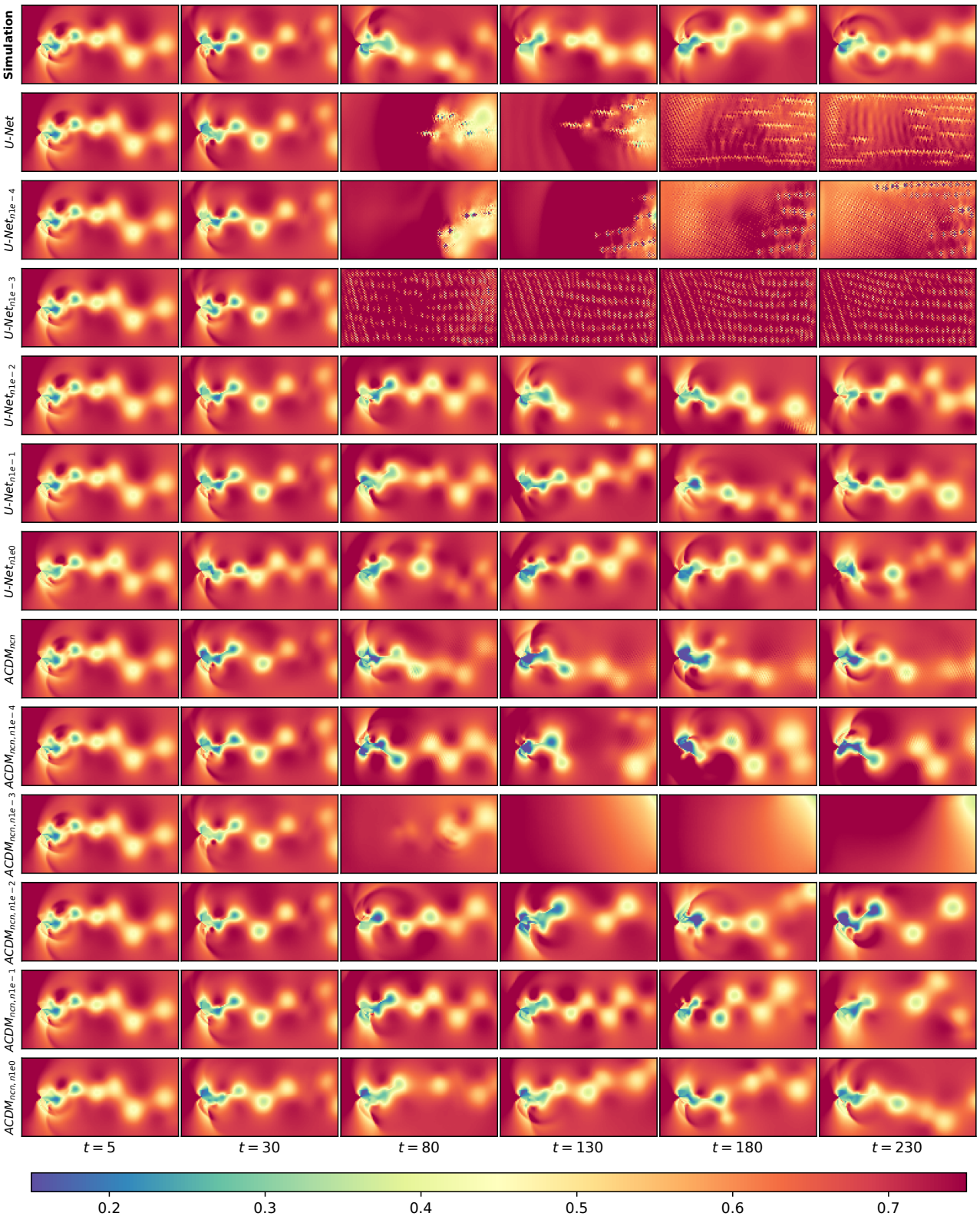

Figure 40. Training Noise Ablation (see App. I.3): Pressure predictions from $\texttt{Tra}_{\texttt{long}}$.

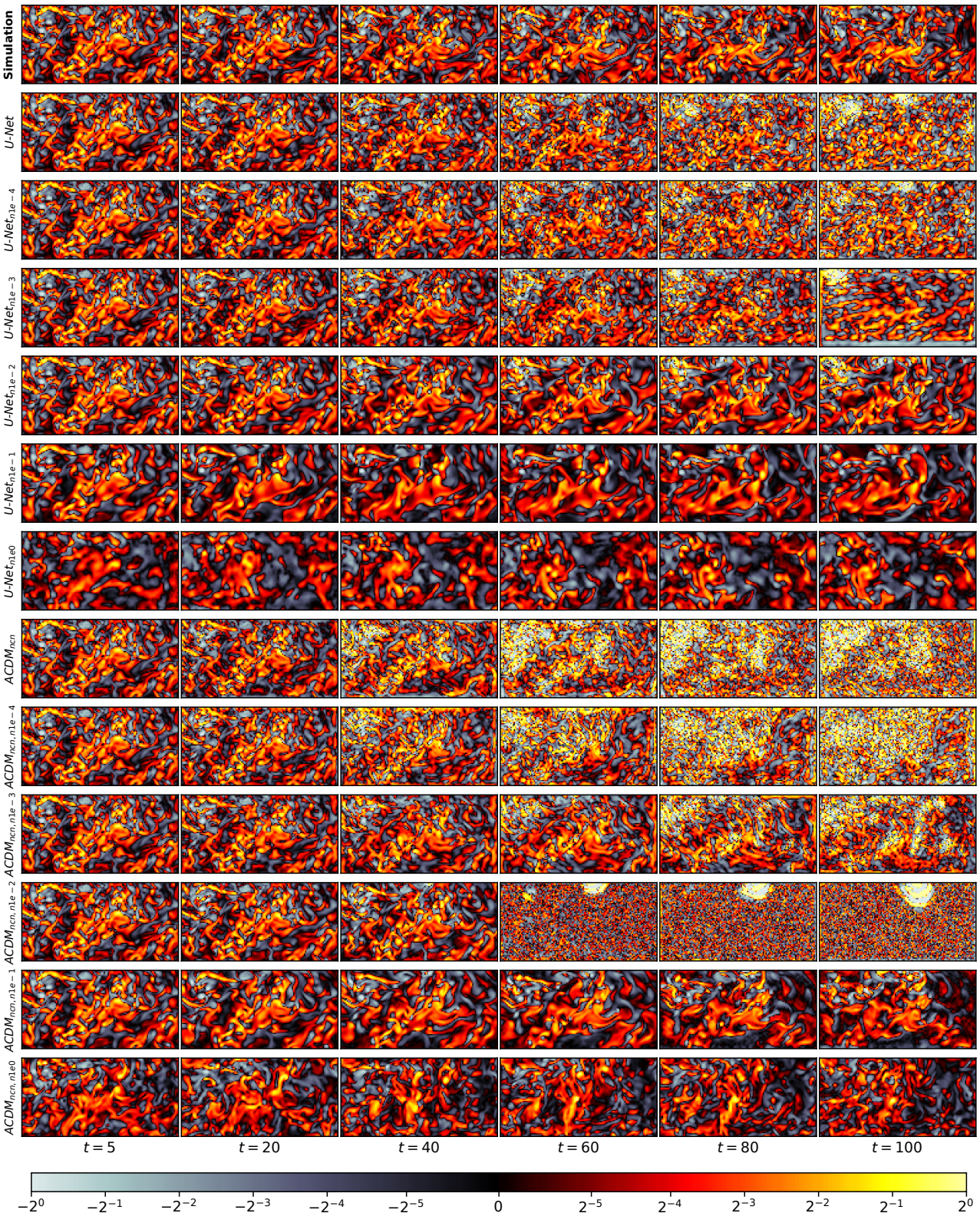

Figure 41. Training Noise Ablation (see App. I.3): Vorticity predictions from `Iso`.

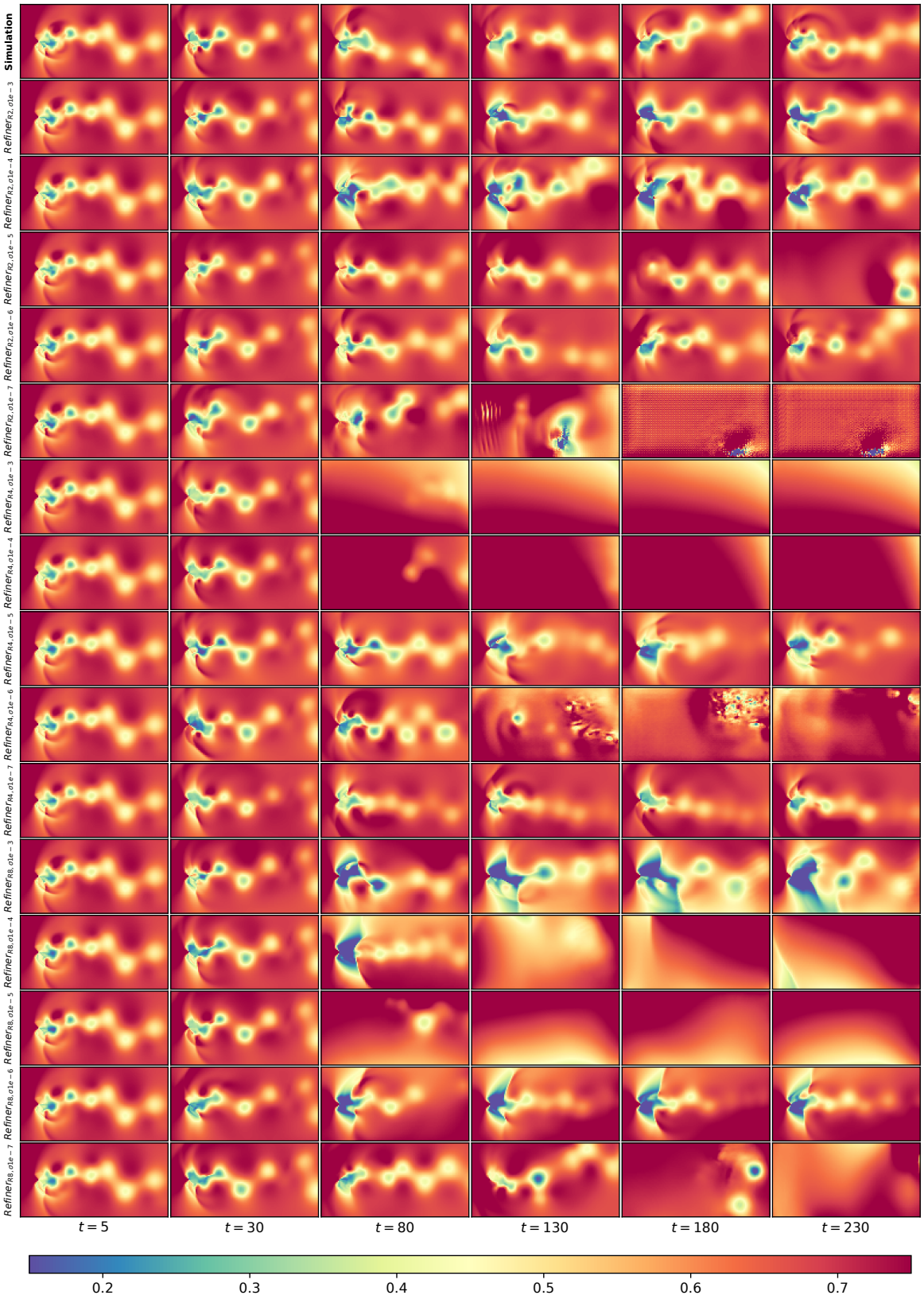

Figure 42. Comparison to PDE-Refiner (see App. I.6): Pressure predictions from $\texttt{Tra}_{\texttt{long}}$.

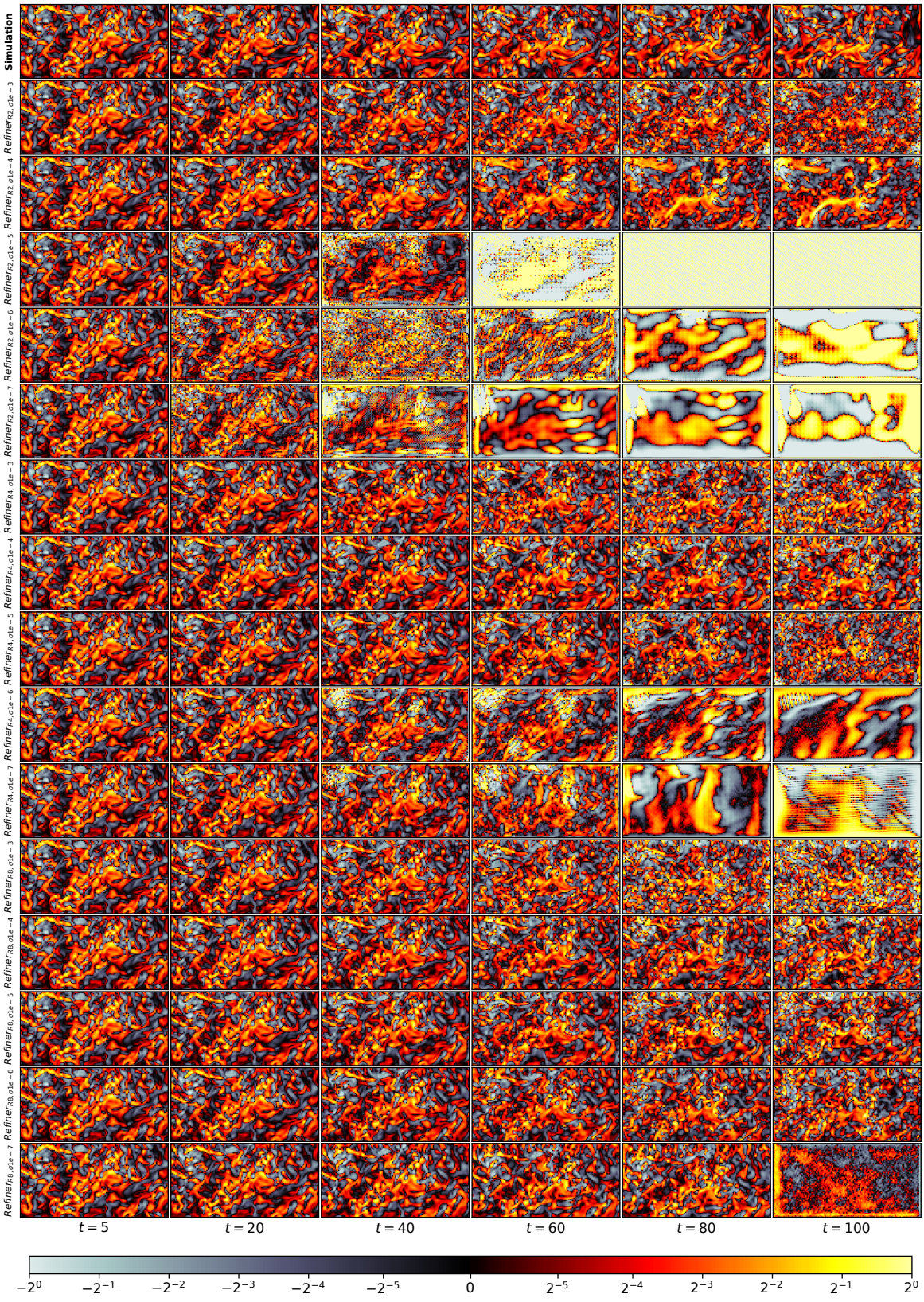

Figure 43. Comparison to PDE-Refiner (see App. I.6): Vorticity predictions from `Iso`.