TURBULENT FLOW SIMULATION USING AUTOREGRESSIVE CONDITIONAL DIFFUSION MODELS

Georg Kohl, Li-Wei Chen, & Nils Thuerey
Technical University of Munich
Munich, Germany
{georg.kohl, liwei.chen, nils.thuerey}@tum.de

ABSTRACT

Simulating turbulent flows is crucial for a wide range of applications, and machine learning-based solvers are gaining increasing relevance. However, achieving stability when generalizing to longer rollout horizons remains a persistent challenge for learned PDE solvers. We address this challenge by introducing a fully data-driven fluid solver that utilizes an autoregressive rollout based on conditional diffusion models. We show that this approach offers clear advantages in terms of rollout stability compared to other learned baselines. Remarkably, these improvements in stability are achieved without compromising the quality of generated samples, and our model successfully generalizes to flow parameters beyond the training regime. Additionally, the probabilistic nature of the diffusion approach allows for inferring predictions that align with the statistics of the underlying physics. We quantitatively and qualitatively evaluate the performance of our method on a range of challenging scenarios, including incompressible and transonic flows, as well as isotropic turbulence.

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 methods. These approaches have shown considerable promise in terms of enhancing the accuracy and efficiency of fluid simulations (Wiewel et al., 2019; Han et al., 2021; Geneva & Zabaras, 2022; Stachenfeld et al., 2022).

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 (Um et al., 2020; Kochkov et al., 2021). 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 address these issues by exploring the usefulness of the recently emerging conditional diffusion models (Ho et al., 2020; Song et al., 2021b) for turbulent flows serving as representatives for 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: *Does the increased inference cost of autoregressive diffusion models pay off in terms of posterior sampling, rollout stability, and accuracy for fluid simulations?* Our focus on fluid flows makes it possible to analyze the generated posterior samples with the statistical temporal metrics established by turbulence research (Dryden, 1943). Unlike application areas like imaging or speech, where the exact distribution of

Our source code is available at https://github.com/tum-pbs/autoreg-pde-diffusion.

possible solutions is typically unknown, these turbulence metrics make it possible to reliably evaluate the quality of different samples generated by a probabilistic model. To summarize, the central contributions of our work are as follows: (*i*) We propose to use a conditional diffusion approach with an autoregressive rollout to produce a probabilistic surrogate simulator. This approach is robust, can be flexibly conditioned on flow parameters, and generalizes to parameters outside the training domain. (*ii*) Most notably, we show that using a diffusion-based approach provides significant benefits over common learned autoregressive flow simulation methods, especially in terms of accuracy and stability over time. (*iii*) We additionally demonstrate that this simulator can generate physically plausible posterior samples, the statistics of which match those of the underlying ground truth physics.

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; Um et al., 2020; List et al., 2022) to increase simulation accuracy. In addition, differentiable solvers have been applied to solve inverse problems such as fluid control (Holl et al., 2020). An overview can be found, e.g., in Thuerey et al. (2021). When the solver is not integrated into the computational graph, typically a surrogate model is trained with pre-computed data 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; Wu et al., 2022) 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 a reduced-order model (Hemmasian & Farimani, 2023), for example to simulate incompressible flows via Koopman-based latent dynamics (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), Bayesian neural networks for uncertainty quantification over time (Geneva & Zabaras, 2019), or custom multi-scale architectures (Wang et al., 2020). Furthermore, various works utilize message passing architectures (Pfaff et al., 2021; Brandstetter et al., 2022), and adding noise to training inputs was likewise proposed to improve temporal prediction stability for graph networks (Sanchez-Gonzalez et al., 2020). Han et al. (2021) combine a transformer-based latent model with a graph network encoder and decoder.

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). In addition, generative hybrid approaches were proposed, for instance diffusion autoencoders (Preechakul et al., 2022) or score-based latent models (Vahdat et al., 2021).

Conditional Diffusion Algorithms Diffusion models for conditional image generation are typically conditioned on simple class labels (Dhariwal & Nichol, 2021) or textual inputs (Saharia et al., 2022). Ho & Salimans (2022) proposed a classifier-free diffusion guidance approach, while Yang & Mandt (2022) condition a diffusion model on an autoencoder latent space for image compression. Pre-trained diffusion models were also 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). Song et al. (2022) combine an unconditional diffusion model with inverse problem solving for medical applications. For an in-depth review of diffusion approaches we refer to Yang et al. (2022).

Diffusion Models 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. 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. Very few works exist that apply diffusion methods to transient physical processes. Holzschuh et al. (2023) utilize score matching to solve inverse problems, while Shu et al. (2023) employ a physics-informed diffusion model for a frame-by-frame super-resolution task. Lienen et al. (2023) take early steps towards turbulent flows in 3D, via a purely generative diffusion setup based on boundary geometry information. In contrast to our work, Yang & Sommer (2023) condition a diffusion model on the

absolute physical time similar to the diffusion time step for fluid field prediction, and correspondingly report difficulties with temporal stability and unphysical predictions.

3 Autoregressive Conditional Diffusion

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 spatial fields, like velocity, and scalar parameters such as the Reynolds number. Numerical solvers f iteratively predict $s^t = f(s^{t-1})$, and we similarly propose to use a diffusion model f_θ with parameters θ to autoregressively predict $s^t \sim f_\theta(s^{t-1})$. Below, we briefly summarize the basics of diffusion models, before providing details on how to condition and unroll them to obtain stable temporal predictions of physics systems. It is important to distinguish the *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* in the Markov chain, denoted by $r \in 0, 1, \ldots, R$ subscripts.

3.1 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). 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 that is similar to the distribution of the training data. Note that the latent space of a DDPM by construction has the same dimensionality as the input space, in contrast to, e.g., variational autoencoders (VAEs) (Kingma & Welling, 2014). Thereby, it avoids problems with the generation of high frequency details due to compressed representations. Compared to generative adversarial networks (GANs), diffusion models typically do not suffer from mode collapse problems or convergence issues (Metz et al., 2017).

3.2 Conditioning

Practical physics simulators need to be conditioned on information like the initial state and characteristic dimensionless quantities. To obtain a conditional DDPM, 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 task at inference time is the conditional prediction $P(\boldsymbol{d}_0|\boldsymbol{c}_0)$. During training, the basic DDPM algorithm remains unchanged as $\boldsymbol{x}_r = (\boldsymbol{c}_r, \boldsymbol{d}_r)$, with $\boldsymbol{c}_r \sim q(\,\cdot\,|\boldsymbol{c}_{r-1})$ and $\boldsymbol{d}_r \sim q(\,\cdot\,|\boldsymbol{d}_{r-1})$, 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.,

$$\boldsymbol{x}_r = (\boldsymbol{c}_r, \boldsymbol{d}_r) \text{ where } \boldsymbol{c}_r \sim q(\,\cdot\,|\boldsymbol{c}_{r-1}) \text{ and } \boldsymbol{d}_r \sim p_\theta(\,\cdot\,|\boldsymbol{x}_{r+1}). \tag{3}$$

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. This conditioning technique is illustrated in Fig. 1.

3.3 Training and Autoregressive Rollout

To extend the conditional setup to temporal tasks, we build on autoregressive single-step prediction as outlined above, with k previous steps: $s^t \sim f_\theta(\,\cdot\,|s^{t-k}, \ldots, s^{t-1})$. The conditional DDPM for 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 $\boldsymbol{d}_0$.

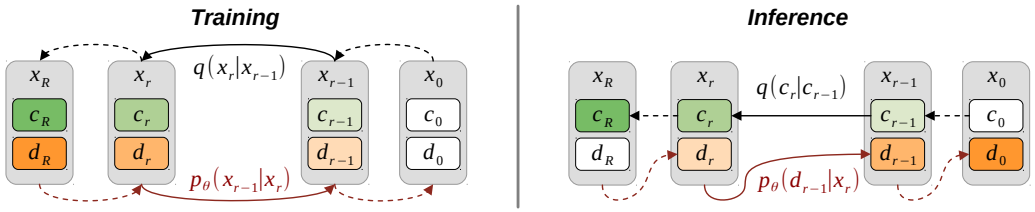

Figure 1: Diffusion conditioning approach with the forward (black) and reverse process (red) during training and inference. White backgrounds for $\boldsymbol{c}$ or $\boldsymbol{d}$ indicate given information, i.e., inputs in each phase, and $\boldsymbol{d}_0$ is the generated result during inference, i.e., the prediction of the next simulation step.

This state consists of dense simulation fields like velocity or pressure, and scalar parameters like the Mach number (see right of Fig. 2). The corresponding conditioning consists of k previous simulation states $\boldsymbol{c}_0 = (s^{t-k}, \dots, s^{t-1})$. 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).

This training objective 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. (3) 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 $\boldsymbol{c}_0^t = (s^{t-k}, \dots, s^{t-1})$. Then, the predicted next time step is $s^t = \boldsymbol{d}_0^t$. This process is visualized in Fig. 2, and we denote models trained with this approach as *autoregressive conditional diffusion models* (ACDMs) in the following.

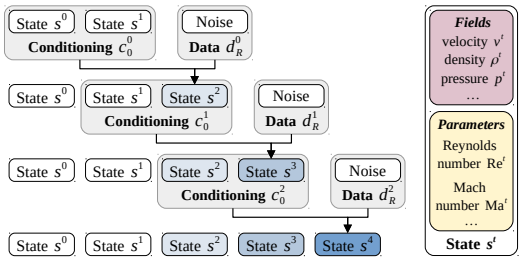

Figure 2: Autoregressive simulation rollout for $k = 2$ (left) and simulation state contents (right).

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. Furthermore, this autoregressive rollout approach 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, as proposed by 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 potentially be achieved via tricks such as fixed temporal and spatial noise patterns.

3.4 Implementation

We employ a widely used U-Net (based on Ronneberger et al., 2015) with various established smaller architecture modernizations (Ho et al., 2020; Dhariwal & Nichol, 2021), to learn the reverse process with a linear variance schedule. We use $k = 2$ previous steps for the model input. Unlike the original DDPM, we achieve high-quality samples with as little as $R = 20$ diffusion steps. We believe this stems from our strongly conditioned setting, in line with other recent works that achieve competitive results with $R \approx 30$ (Karras et al., 2022; Chung et al., 2022). 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. Likewise, scalar simulation parameters are concatenated as constant, spatially expanded channels. The source code for this work is available at `https://github.com/tum-pbs/autoreg-pde-diffusion`.

3.5 Baseline Models

The success of transformer architectures (Vaswani et al., 2017) and their recent application to physics predictions (Han et al., 2021) raises the question how the ACDM approach fares in comparison to state-of-the-art transformer architectures. Being tailored to sequential processing with a long term observation horizon, these models work on a latent space with a reduced size. In contrast, diffusion models by construction need to operate on the full spatial resolution. In addition, a crucial question is how much difference the diffusion training itself makes in comparison to a classic supervised training approach. Hence, in the following we compare against several transformer-based architectures as representatives of fundamentally different latent-space predictors, and a network with identical architecture to the ACDM model that is trained with a classic supervised approach instead of a diffusion-based one. 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 VAE, denoted by TF_{VAE}. All transformer architectures have access to a large number of previous steps and use a rollout schedule in line with Han et al. (2021) during training, however teacher forcing is removed. By default we use a 30 step input window and a rollout length of 60. The classic supervised training variant employs the same U-Net architecture as the ACDM model, and is trained with an MSE loss to predict one future state with a single pass. It is denoted by *U-Net* in the following. 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.

4 Experiments

We quantitatively and qualitatively evaluate our DDPM-based simulator and the baseline models on three 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 in the form of simulation parameters outside of the training data range. Figure 3 shows vorticity visualizations of example sequences. For the first two experiments we use $R = 20$, and for the third $R = 50$ diffusion steps. Further details for each experiment are provided in Appendix A.

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 the generalization on the extrapolation test sets $\texttt{Inc}_{\texttt{low}}$ with $Re \in [100, 180]$ 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 of 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. These flows require the simulation of a varying density, and exhibit the formation of shock waves that interact with the flow, especially at higher Mach numbers. 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 the Mach number 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 using $\texttt{Tra}_{\texttt{ext}}$ with $Ma \in [0.50, 0.52]$, 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

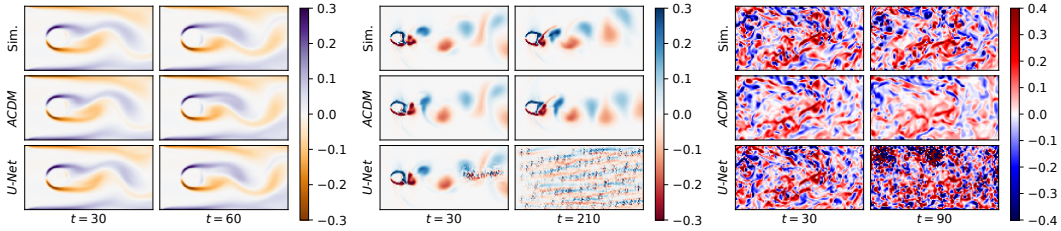

Figure 3: Examples from $\texttt{Inc}_{\texttt{low}}$ with $Re = 120$ (left), $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$ (middle), and $\texttt{Iso}$ with $z = 280$ (right). Shown are the states produced by a traditional numerical solver at the top, and predictions by *ACDM* and *U-Net* (with identical network structures) below.

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. Especially so for deterministic methods, motivating the choice for a probabilistic approach. 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 $T = 100$.

5 Results

In the following, we analyze the *ACDM* posterior sampling and evaluate the different methods in terms of their general accuracy and temporal stability. Appendix C contains 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 $\texttt{Iso}$, and three for $\texttt{Inc}$ and $\texttt{Tra}$. For the probabilistic methods TF_{VAE} and *ACDM*, five random model evaluations are taken into account per trained model.

5.1 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). For both metrics, lower values indicate better reconstruction accuracy. Reported errors are rollout errors, i.e., computed per time step and field, and averaged over the full temporal rollout.

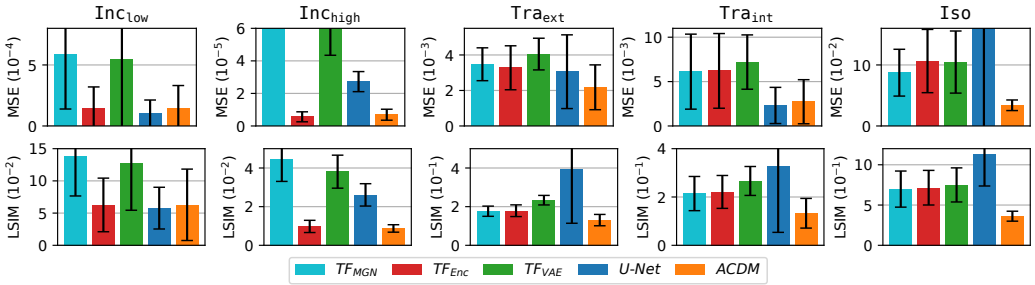

Figure 4: Quantitative comparison across our test sets for different network architectures in terms of MSE (top) and LSiM (bottom). The full, numerical accuracy results are also included in Appendix C.4.

As shown in Fig. 4, *ACDM* produces accurate results with low standard deviation across all experiments and test sets, and consistently outperforms TF_{MGN}. For the easiest test case $\texttt{Inc}$, TF_{Enc} and *U-Net* achieve comparable performance to *ACDM*. On $\texttt{Tra}$, TF_{MGN} and TF_{Enc} perform very similar, but consistently worse than *ACDM*. Both methods severely struggle for $\texttt{Iso}$, as the compressed latent representations are unable to capture the high frequency details of this data set. The probabilistic

variant TF_{VAE}, that shares the same transformer latent model as TF_{Enc}, further reduces accuracy across all tests. The regular *U-Net*, despite its network structure being identical to *ACDM*, frequently performs worse on our more difficult tasks, especially on $\texttt{Tra}_{\texttt{ext}}$ and `Iso`. Prediction errors of *U-Net* are also clearly visible in Fig. 3, where longer rollouts or the strongly underdetermined setting of `Iso` cause diverged solutions. Overall, *ACDM* achieves results comparable with the baselines on the simplest case `Inc`, and outperforms the other methods on the more complex `Tra` data set in almost all evaluations. It even achieves an improvement of close to 50% on the isotropic turbulence case, our most challenging experiment.

5.2 Posterior Sampling

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 produced by the *ACDM* models.

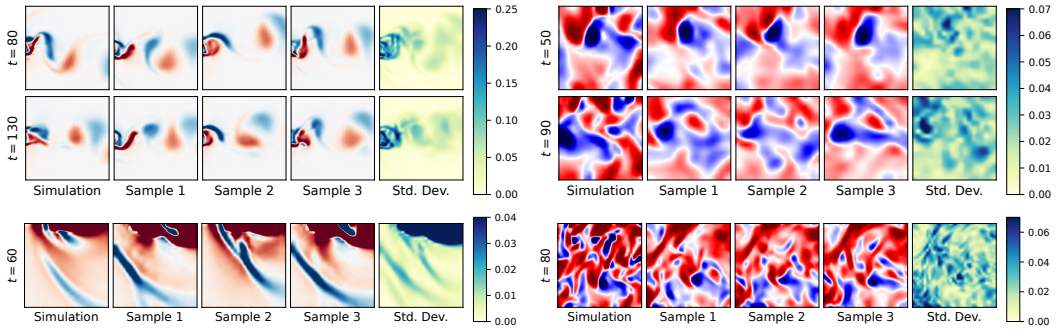

Figure 5: Zoomed *ACDM* posterior samples with corresponding standard deviation from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$ (left), and from `Iso` with $z = 300$ (right) at different time steps t.

First, it becomes apparent that sampling the *ACDM* can produce samples with substantial differences. This is illustrated in Fig. 5, where zoomed areas from three random *ACDM* predictions on $\texttt{Tra}_{\texttt{long}}$ and `Iso` at different time steps are displayed along with a sample from the regular simulation. The diffusion approach produces realistic, diverse features, such as the strongly varying formation of shock waves near the immersed cylinder for $\texttt{Tra}_{\texttt{long}}$. Figure 5 also visualizes the resulting spatially varying standard deviation across five samples. Due to the chaotic nature of both test cases, it increases over time, as expected. The locations of high variance match areas that are more difficult to predict, such as vortices and shock wave regions for $\texttt{Tra}_{\texttt{long}}$, and regions with high vorticity for `Iso`. In contrast, the posterior sampling of TF_{VAE} does not manage to produce such results, as different samples from one TF_{VAE} model exhibit only minor differences. Furthermore, high-frequency structures are either missing or unphysical, e.g., in areas where shock waves should normally occur. A series of example samples is provided in Appendix C.3.

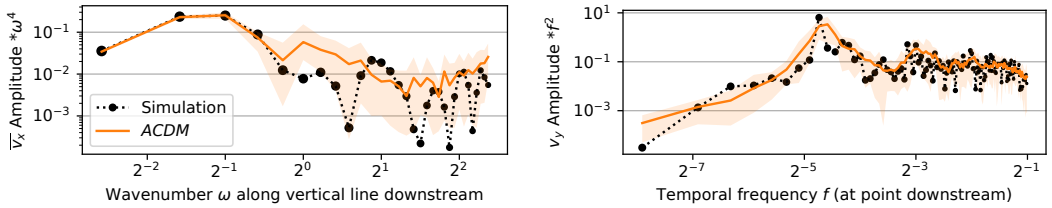

Figure 6: Spatial (left) and temporal (right) frequency analysis across posterior samples for a full sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$. The shaded area shows the 5^{th} to 95^{th} percentile across all trained models and posterior samples.

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 setups allow for using temporal and spatial evaluations to measure whether different samples *statistically* match the reference simulation, as established by turbulence research (Dryden, 1943). While a broader evaluation is provided in Appendix C.1, two metrics for a sequence from $\texttt{Tra}_{\texttt{long}}$ are discussed 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. As shown in Fig. 6, the samples produced by *ACDM* accurately match the statistics of the reference simulation. Even the high frequency content on the right side of the spectrum is on average correctly reproduced by the *ACDM* outputs. Thus, the posterior samples exhibit a high correspondence to the statistical physical behavior of the reference simulations.

5.3 Comparing Temporal Statistics

Temporal statistics as discussed in Sec. 5.2 also highlight the differences of the model architectures under consideration. Figure 7 shows an evaluation in terms of the frequency of the velocity x-component for Iso averaged across every spatial point. All methods based on the transformer architectures are lacking across the frequency band, while *U-Net* clearly overshoots. High temporal frequencies are modeled well by *ACDM*, but it also slightly deviates in terms of lower frequencies. This is most likely caused by the strongly underdetermined nature of the isotropic turbulence experiment, which causes *ACDM* to dissipate spatial high-frequency motions more than necessary. Nonetheless, it still outperforms the other approaches in this setting.

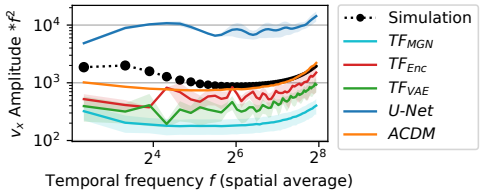

Figure 7: Temporal frequency on a sequence from Iso with $z = 300$.

5.4 Temporal Stability

A central motivation for employing 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 not uncommon. We now evaluate this aspect of the proposed algorithm in more detail, first, by measuring the Pearson correlation coefficient (Pearson, 1920) between prediction and reference over time. We evaluate this for the $\texttt{Inc}_{\texttt{var}}$ test, which contains $T = 250$ steps with a previously unseen change of the Reynolds number over the rollout. Initially, the *U-Net* model is accurate, but it exhibits a faster decorrelation over time, especially compared to *ACDM* and TF_{Enc}. This already indicates a lack of tolerance to rollout errors observed on our more complex cases. The transformer variants fare better, and especially TF_{Enc} keeps a high level of correlation that even slightly outpeforms *ACDM*. However, TF_{Enc} has the advantage that it receives $k = 30$ previous simulation states during training and inference to predict the next state. While TF_{VAE} also performs well here, it exhibits problems with temporal coherence as its consecutive output steps can vary significantly on more complex cases, as shown in more detail in Appendix C.3. *ACDM* keeps the high level of correlation with the reference solution requiring only $k = 2$ previous states.

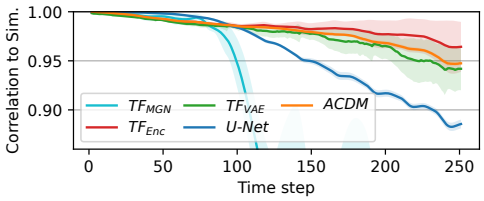

Figure 8: Correlation of predictions from different methods with the reference simulation over the rollout on $\texttt{Inc}_{\texttt{var}}$.

To assess the dynamics of more complex cases, we measure the magnitude of the rate of change of s, computed as $\|(s^t - s^{t-1})/\Delta t\|_1$ for every normalized time step. This metric stays meaningful even for long rollout times, and indicates whether a solver preserves the expected evolution of states as given by the reference simulation. Figure 9 shows this evaluation for $\texttt{Tra}_{\texttt{long}}$ and Iso. For $\texttt{Tra}_{\texttt{long}}$, the reference simulation features steady oscillations as given by the main vortex shedding frequency. All transformer variants, most likely due to the temporal updates being performed purely in latent

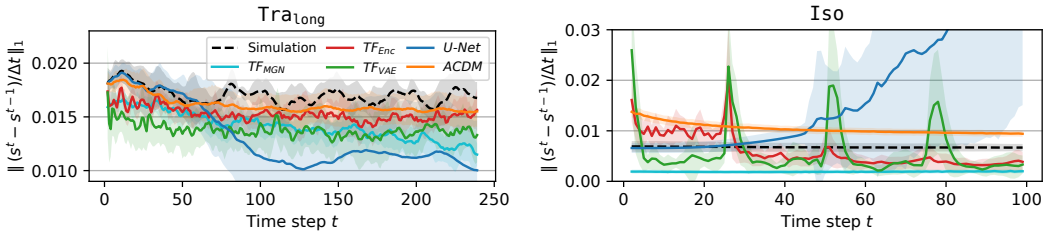

Figure 9: Stability analysis via error to previous time step on $\texttt{Tra}_{\texttt{long}}$ (left) and `Iso` (right).

space, exhibit a relatively constant rate of change that undershoots in comparison to the reference and is temporally slightly inconsistent. The *ACDM* simulator closely follows the reference until ca. $t = 60$, and then likewise transitions to a relatively constant rate of change. However, in contrast to the transformer models, it is able to better maintain the average rate as prescribed by the reference simulation. The right side of Fig. 9 repeats this evaluation for the forced isotropic turbulence case. Its isotropic nature in combination with the forcing leads to an almost constant rate of change over time for the reference simulation. While all methods show a slight decay and are struggling to follow the reference, *TF$_{MGN}$* immediately collapses and barely produces meaningful temporal behavior. *TF$_{Enc}$* and *TF$_{VAE}$* do better, but nonetheless fall off too quickly, and exhibit undesirable spikes corresponding to their temporal prediction window of 25 previous steps at training and inference time. Despite the slight decay, *ACDM* fares best among these methods, and closely matches the reference simulation, while still being slightly too dissipative. In both cases, *U-Net* is initially most accurate, but diverges quickly and significantly from the reference. These results confirm the initial hypothesis, and show that the DDPM-based training not only leads to better instantaneous predictions, but also yields an excellent temporal stability.

Discussion So far, we have evaluated *ACDM* and *U-Net* with training methodologies that were kept as similar as possible for fairness. However, several works have reported improvements from unrolling predictions at training time (Lusch et al., 2018; Geneva & Zabaras, 2020; Um et al., 2020; Kochkov et al., 2021). While this is not feasible for the current training approaches of diffusion models, it can be applied to the supervised training of the *U-Net* architecture. We denote the number of states used for unrolling at training time by m. This means, a network with $m = 4$ receives s^{t-1} as the input and predicts a 3 step rollout of (s^t, s^{t+1}, s^{t+2}) during training with a corresponding loss. The gradient is backpropagated through all steps in each training iteration, and the number of input steps k is set to one for these variants. Memory and computational resources of this training approach scale almost linearly with m.

This additional complexity at training time improves the results of the *U-Net* when m is chosen correctly. In most cases, up to $m = 8$ unrolling steps improve the behavior of the model. However, longer unrolling at training time with $m = 16$ showed first signs of recurrent instabilities, and led to a deterioration in quality (for instance, we observed this when training on `Iso`). These instabilities could potentially be addressed by a training schedule which increases m over time. However, this would add further hyperparameters to the training algorithm. Nonetheless, this modification of the training yields *U-Net* models that significantly improve in terms of stability and accuracy, show no exploding behavior, and approach the overall quality of the *ACDM* models. Naturally, this does not provide the *U-Net* with capabilities for posterior sampling. Considering the `Iso` test set as an example, the MSE of *U-Net* improves from 0.258 for the regular version to 0.037 for $m = 4$. On the other hand, $m = 8$ results in an MSE of 0.045, while the *ACDM* network trained without unrolling achieves 0.034. These experiments point to an interesting direction for future work: the inclusion of autoregressive unrolling steps in addition to the diffusion-based iterations at training time. The flow of information through multiple time steps of the physics process during training could also yield positive training signals for autoregressive diffusion models.

6 Limitations and Conclusions

We demonstrated the attractiveness of autoregressive conditional diffusion models for the simulation of complex flow phenomena. Our findings show that using a diffusion-based approach has significant

advantages in terms of rollout stability, while at the same time enabling probabilistic inference. Our results show a remarkable temporal stability of the diffusion-based training, which surpasses the established methodologies for classical supervised training and recent advances from sequence modeling in the form of transformers. The different posterior samples generated by our method faithfully reproduce the physical statistics of the reference solutions.

This flexibility of the learned simulator comes at higher computational cost for inference, as each step of the simulated rollout requires multiple iterations for the diffusion rollout. Evaluated on a single NVIDIA RTX A5000 GPU over a sequence with $T = 1000$, *U-Net* can on average produce one time step in about 0.0129 seconds, while *ACDM* with $R = 20$ requires about 0.202 seconds. We believe that recent advances in improved sampling procedures for diffusion models such as distillation (Salimans & Ho, 2022) are a promising avenue for improved performance in future work. However, the advantages in terms of stability of diffusion-based approaches most likely stem from its iterative nature over the diffusion rollout, and hence we anticipate that a constant factor over deterministic, single-pass inference will remain in place.

Naturally, an extension to other PDEs, and to larger, three-dimensional flows is likewise a highly interesting direction. In the latter setting, the excellent temporal stability of the single-step ACDM training will be particularly attractive, as a temporal rollout of three-dimensional data (Sirignano et al., 2020; Um et al., 2020) would quickly incur a huge cost at training time.

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

Filipe de Avila Belbute-Peres, Thomas D. Economon, and J. Zico Kolter. 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.

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. 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.

Georgios Batzolis, Jan Stanczuk, Carola-Bibiane Schönlieb, and Christian Etmann. Conditional image generation with score-based diffusion models. *arXiv*, 2021. doi:10.48550/arXiv.2111.13606.

Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. *arXiv*, 2023. doi:10.48550/arXiv.2304.08818.

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

Hyungjin Chung, Byeongsu Sim, and Jong Chul Ye. Come-closer-diffuse-faster: Accelerating conditional diffusion models for inverse problems through stochastic contraction. In *2022 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12403–12412, 2022. doi:10.1109/CVPR52688.2022.01209.

Indu Kant Deo, Rui Gao, and Rajeev Jaiman. 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.

Prafulla Dhariwal and Alexander Quinn Nichol. 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.

Hugh L. Dryden. 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.

Thomas Economon, Francisco Palacios, Sean Copeland, Trent Lukaczyk, and Juan Alonso. Su2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54:1–19, 2015. doi:10.2514/1.J053813.

Nicholas Geneva and Nicholas Zabaras. Quantifying model form uncertainty in reynolds-averaged turbulence models with bayesian deep neural networks. *Journal of Computational Physics*, 383: 125–147, 2019. ISSN 0021-9991. doi:10.1016/j.jcp.2019.01.021.

Nicholas Geneva and Nicholas Zabaras. 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.

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

Xu Han, Han Gao, Tobias Pfaff, Jian-Xun Wang, and Liping Liu. 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.

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

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv*, 2022. doi:10.48550/arXiv.2207.12598.

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

Jonathan Ho, Tim Salimans, Alexey A. Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. 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.

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

Benjamin Holzschuh, Simona Vegetti, and Nils Thuerey. Score matching via differentiable physics. *arXiv*, 2023. doi:10.48550/arXiv.2301.10250.

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

Tianli Hu and Shijun Liao. 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.

Aapo Hyvärinen. 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.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems 35*, 2022. URL https://openreview.net/forum?id=k7FuTOWMOc7.

Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. 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.

Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus H. Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70, 2019. doi:10.1111/cgf.13619.

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

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

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

Georg Kohl, Kiwon Um, and Nils Thuerey. 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.

Marten Lienen, Jan Hansen-Palmus, David Lüdke, and Stephan Günnemann. Generative diffusion for 3d turbulent flows. *arXiv*, 2023. doi:10.48550/arXiv.2306.01776.

Björn List, Li-Wei Chen, and Nils Thuerey. 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.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 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.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. 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.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. In *5th International Conference on Learning Representations (ICLR 2017)*, 2017. URL https://openreview.net/forum?id=BydrOIcle.

Parviz Moin and Krishnan Mahesh. 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.

Alexander Quinn Nichol and Prafulla Dhariwal. 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.

Mette S Olufsen, Charles S Peskin, Won Yong Kim, Erik M Pedersen, Ali Nadim, and Jesper Larsen. 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.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, 2019. doi:10.48550/arXiv.1912.01703.

Karl Pearson. Notes on the history of correlation. *Biometrika*, 13(1):25–45, 1920. doi:10.1093/biomet/13.1.25.

Eric Perlman, Randal Burns, Yi Li, and Charles Meneveau. 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.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. 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.

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

Konpat Preechakul, Nattanat Chatthee, Suttisak Wizadwongsa, and Supasorn Suwajanakorn. Diffusion autoencoders: Toward a meaningful and decodable representation. In *2022 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10609–10619, 2022. doi:10.1109/CVPR52688.2022.01036.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *Lecture Notes in Computer Science*, pp. 234–241, 2015. doi:10.1007/978-3-319-24574-4_28.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. 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.

Tim Salimans and Jonathan Ho. 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.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. 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.

Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. 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.

Dule Shu, Zijie Li, and Amir Barati Farimani. 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.

Justin A. Sirignano, Jonathan F. MacArt, and Jonathan B. Freund. 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.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. 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.

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

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 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.

Yang Song, Liyue Shen, Lei Xing, and Stefano Ermon. 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.

P. R. Spalart, S. Deck, M. L. Shur, K. D. Squires, M. Kh. Strelets, and A. Travin. 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.

Kimberly L. Stachenfeld, Drummond Buschman Fielding, Dmitrii Kochkov, Miles D. Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter W. Battaglia, and Alvaro Sanchez-Gonzalez. 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.

Nils Thuerey, Philipp Holl, Maximilian Mueller, Patrick Schnell, Felix Trost, and Kiwon Um. *Physics-based Deep Learning*. WWW, 2021. URL https://physicsbaseddeeplearning.org.

Kiwon Um, Robert Brand, Yun Fei, Philipp Holl, and Nils Thuerey. 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.

Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. In *Advances in Neural Information Processing Systems 34*, pp. 11287–11302, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5dca4c6b9e244d24a30b4c45601d9720-Abstract.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 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.

Siddhartha Verma, Guido Novati, 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.

Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. 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.

S. Wiewel, M. Becher, and N. Thuerey. 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.

Tailin Wu, Takashi Maruyama, and Jure Leskovec. Learning to accelerate partial differential equations via latent global evolution. In *Advances in Neural Information Processing Systems 35*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/0f817dcbad81afb21fb695f1b2e55e44-Abstract-Conference.html.

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

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

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

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Yingxia Shao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *arXiv*, 2022. doi:10.48550/arXiv.2209.00796.

Ruihan Yang and Stephan Mandt. Lossy image compression with conditional diffusion models. *arXiv*, 2022. doi:10.48550/arXiv.2209.06950.

A Data Details

In the following, we provide details for each simulation setup: the incompressible wake flow `Inc` in Appendix A.1, the transonic cylinder flow `Tra` in Appendix A.2, and the isotropic turbulence `Iso` in Appendix A.3.

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.

Our domain setup is illustrated in Fig. 10. 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 models on a reduced resolution of 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.

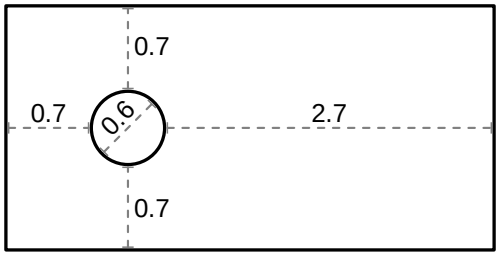

Figure 10: Simulation domain for incompressible flow simulation.

We generated a data set of 91 sequences with Reynolds number $Re \in \{100, 110, \ldots, 990, 1000\}$. 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 $Re \in \{200, 210, \ldots, 890, 900\}$ for $t \in [800, 1300]$. Training and test sequences use a temporal stride of 2. As test sets we use:

- $\texttt{Inc}_{\texttt{low}}$: five sequences with $Re \in \{100, 120, 140, 160, 180\}$ for $t \in [1000, 1120)$ with $T = 60$.
- $\texttt{Inc}_{\texttt{high}}$: five sequences with $Re \in \{920, 940, 960, 980, 1000\}$ for $t \in [1000, 1120)$ with $T = 60$.
- $\texttt{Inc}_{\texttt{var}}$: 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 $\texttt{Inc}_{\texttt{var}}$ case, we replace the model predictions of Re that are learned to be constant for *ACDM* and *U-Net* 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 parameter embedding is provided as a first input step for the latent processor.

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

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. 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 simulation for 150000 iterations, and use every 50^{th} step once the vortex street is fully developed after 100000 iterations. This leads to $T = 1000$ exported steps with velocity, density, 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.

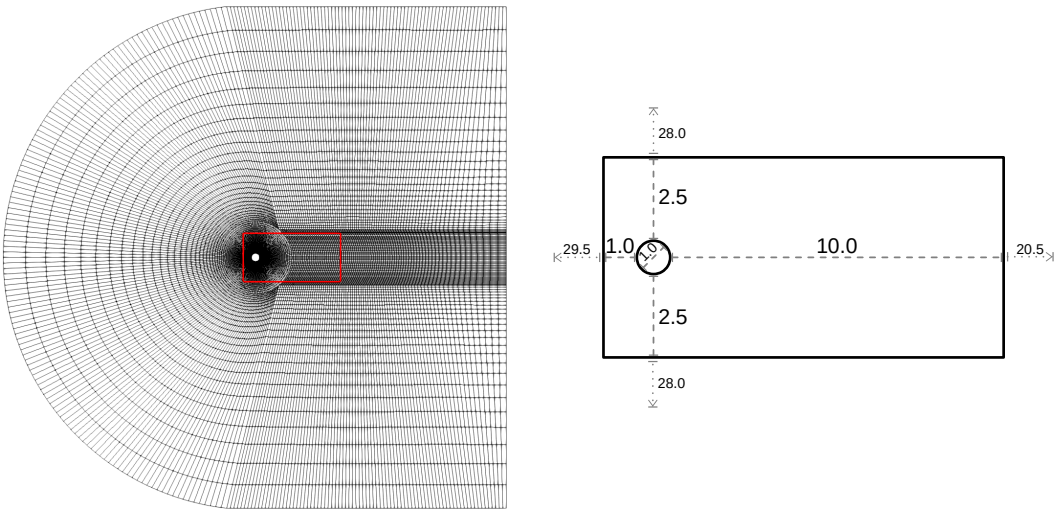

Figure 11: Full simulation mesh with highlighted resampling area (left) and resampling domain setup (right) for the transonic flow simulation.

The computational mesh is illustrated on the left in Fig. 11. Inference is focused on the near field region around the obstacle (marked in red on the left, and shown in detail on the right). To interpolate from the original mesh to the resampled 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 Appendix A.1.

We created a data set of 41 sequences with Mach number $Ma \in \{0.5, 0.51, \ldots, 0.89, 0.90\}$ at Reynolds number 10^4 with the $T = 1000$ exported steps each. We sequentially ran the simulations on one node of a CPU cluster 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 during the simulation. All models are trained on the data of 33 sequences with $Ma \in \{0.53, 0.54, \ldots, 0.62, 0.63\} \cup \{0.69, 0.70, \ldots, 0.89, 0.90\}$. Training and test sequences use a temporal stride of 2. The used test cases for this 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 $T = 60$.

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

- $\texttt{Tra}_{\texttt{int}}$: 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 test case, we make use of the 3D *isotropic1024coarse* simulation from the Johns Hopkins Turbulence Database[3] (Perlman et al., 2007). It simulates 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 $(e_x, e_y, e_z, e_t) = (256, 128, z, 1000)$ 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 identical 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 $T = 100$.

B Implementation and Model Details

Using the data generated with the techniques described above, the deep learning methods of this work are implemented in PyTorch (Paszke et al., 2019). Each model is trained and evaluated on a server with an NVIDIA RTX A5000 GPU with 24GB of memory and an Intel Xeon Gold 6242R CPU with 20 cores at 3.1 GHz. For every model we optimize network weights using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 10^{-4}, where the batch size is chosen to fully utilize GPU memory. 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 Appendix 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: *ACDM* in Appendix B.1, *U-Net* in Appendix B.2, and the transformer-based models in Appendix B.3.

B.1 ACDM Implementation

For the *ACDM* models, we follow the established general U-Net network 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. 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,

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

- use a diffusion time embedding for r via a Transformer sinusoidal position embedding (Vaswani et al., 2017) combined with an MLP, that is add to the input of every ConvNeXt block,
- we train the resulting model via reparameterization.

During training, the batch size is 64 and we make use of a Huber loss. Training *ACDM* with $k = 2$ for 3100 epochs on `Inc` took 49 hours per model, and 54 hours on `Tra`. For `Iso` each model was trained for 100 epochs in 65 hours. During inference, one time step can be predicted in about 5.1398 seconds, when evaluated over a sequence with $T = 1000$ including I/O operations and transfers from CPU to GPU. When only considering pure model inference time, each step is completed in 5.1365 seconds.

B.2 Implementation of U-Net (and U-Net Variants)

For the implementation of *U-Net* we use an identical U-Net architecture as described above in Appendix 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 subequent time step.

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. Accordingly, training for 1000 epochs on `Inc` took 19 hours for a basic *U-Net* model, while 55 hours were required for *U-Net*$_{m16}$. On `Tra`, training *U-Net* for 1000 epochs was completed in 18 hours (52 hours for *U-Net*$_{m16}$), while 100 epochs on `Iso` took 90 hours for each *U-Net* model (265 hours for *U-Net*$_{m16}$). During inference, one time step can be predicted in about 0.0129 seconds, when evaluated over a sequence with $T = 1000$ including I/O operations and transfers from CPU to GPU. When only considering pure model inference time, each step is completed in 0.0096 seconds.

B.3 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 $\rightarrow$ 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 size of the latent space. The layers in the encoder 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 and skip connections:

1. Conv($L \rightarrow w_d, 1, 1, 0$) + ReLU + Upsample$(4, 2, nearest)$
2. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Upsample$(2, 2, nearest)$

3. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Upsample($2, 2, nearest$)
4. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Upsample($2, 2, nearest$)
5. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Upsample($2, 2, nearest$)
6. Conv($w_d + L \rightarrow w_d, 3, 1, 1$) + ReLU + Upsample($2, 2, bilinear$)
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 by 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 by 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 of 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 by 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). We do not train the decoder to recover values inside the cylinder area for `Inc` and `Tra`, by applying a binary masking (also see Appendix 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 by 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

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

In terms of computational costs, training these transformer-based models for 5000 epochs on `Inc` took 36-42 hours per model, and 35-45 hours on `Tra`. For `Iso`, each model was trained for 200 epochs in 65-69 hours. During inference, one time step can be predicted in about 0.0046 seconds, when evaluated over a sequence with $T = 1000$ including I/O operations and transfers from CPU to GPU. When only considering pure model inference time, each step is completed in 0.0009 seconds.

C Additional Results and Evaluations

In the following, we provide additional evaluations and results. This includes further frequency evaluations in Appendix C.1, various prediction examples across models and experiments in Appendix C.2, and an analysis of the posterior sampling of *TF$_{VAE}$* in Appendix C.3. Finally, we include

full numerical results for the accuracy analysis and perform model ablations in terms of accuracy in Appendix C.4, and additional temporal stability evaluations in Appendix C.5.

C.1 Additional Frequency Evaluations

In addition to the statistical frequency evaluations in the main paper, here we provide further evaluations on different data sets across the models under consideration. For $\texttt{Inc}_{\texttt{low}}$, we evaluate the wavenumber of the horizontal motion across a vertical line in the flow (averaged over time), shown on the left of Fig. 12. All models and posterior samples for the given sequence are used in the analysis, and the shaded area corresponds to the 5th to 95th percentile across them. For this relatively simple case, all models accurately reconstruct low and medium frequencies. The major difference are high spatial frequencies where all models overshoot to different degrees, however these differences are not apparent to the human eye in prediction visualizations. Nevertheless, *ACDM* remains most accurately in line with the simulation reference.

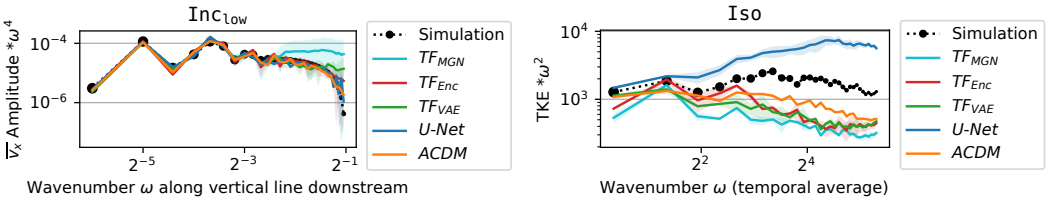

Figure 12: Spatial frequency on a sequence from $\texttt{Inc}_{\texttt{low}}$ with $Re = 100$ (left), and spatial frequency via the turbulent kinetic energy (TKE) on a sequence from $\texttt{Iso}$ with $z = 300$ (right).

On the right in Fig. 12, we compute a spatial frequency analysis on $\texttt{Iso}$ in terms of the turbulent kinetic energy (TKE) averaged across all time steps. All transformer-based methods can only reproduce medium spatial frequencies and lack in terms of low and high frequencies. *U-Net* clearly overshoots across the spectrum, leading to a lacking temporal stability as additional energy is introduced in the simulation. Low and high spatial frequencies are modeled best by *ACDM*, but there is a clear gap to the reference simulation, meaning *ACDM* is more dissipative than necessary in the spatial high-frequency regime. This is most likely caused by the strongly under-determined setting of the isotropic turbulence case, where even a numerical solver in 2D would struggle to provide accurate predictions.

For the frequency evaluations on $\texttt{Tra}_{\texttt{long}}$, we follow the setup from the main paper. Spatial frequencies are evaluated via the horizontal motion across a vertical line in the flow (averaged over time), and temporal frequencies of the vertical motion are computed at a point probe. Figure 13 illustrates the evaluation locations on top of the mean flow of a sequence from $\texttt{Tra}_{\texttt{long}}$, where both the point and line probes are positioned one cylinder diameter downstream. Figure 14 contains frequency analyses on a sequence with $Ma = 0.64$ from $\texttt{Tra}_{\texttt{long}}$ for each model architecture. All trained models and posterior samples are used in this analysis, and the shaded area corresponds to the 5th to 95th percentile across them. As in the evaluation on $\texttt{Iso}$ above, *U-Net* explodes and does not follow the ground truth frequency band. The transformer-based architectures sometimes lack in the high spatial and low temporal frequencies, while *ACDM* reconstructs both spatial and temporal frequencies most accurately among the compared approaches.

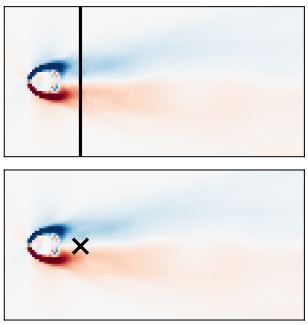

Figure 13: Evaluation line for spatial frequency (top) and evaluation point for temporal frequency analysis (bottom).

C.2 Prediction Examples

Below, we display prediction examples from the analyzed methods. Shown are the different fields contained in an exemplary test sequence from each experiment. Figures 15 and 16 feature the $\texttt{Inc}_{\texttt{var}}$ case, Figs. 17 and 18 contain an example from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$, and Figs. 19 and 20 display a sequence from $\texttt{Iso}$ with $z = 280$.

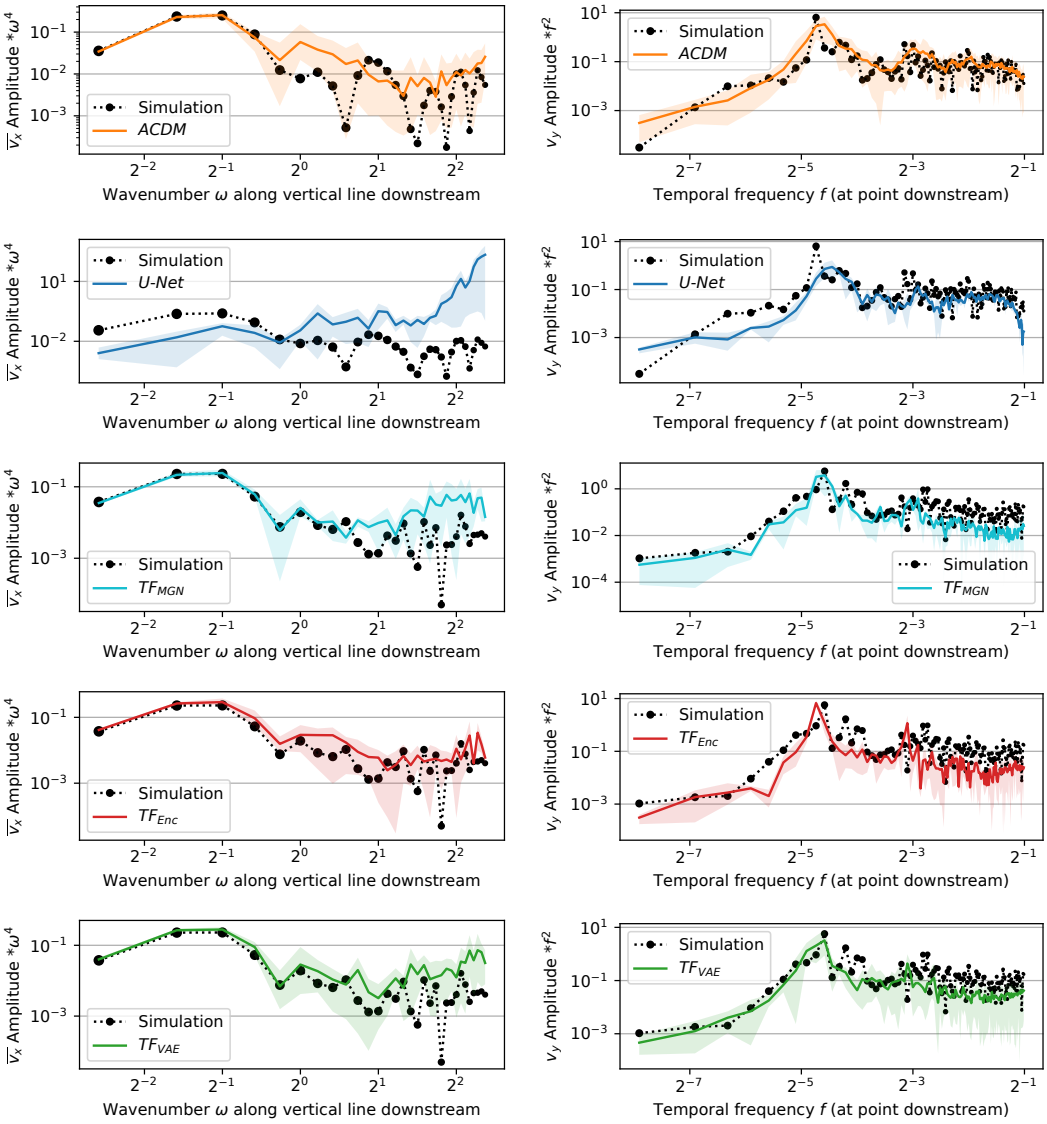

Figure 14: Spatial (left) and temporal (right) frequency analysis for a full sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$. The shaded area shows the 5$^{\text{th}}$ to 95$^{\text{th}}$ percentile across all trained models and posterior samples for probabilistic models.

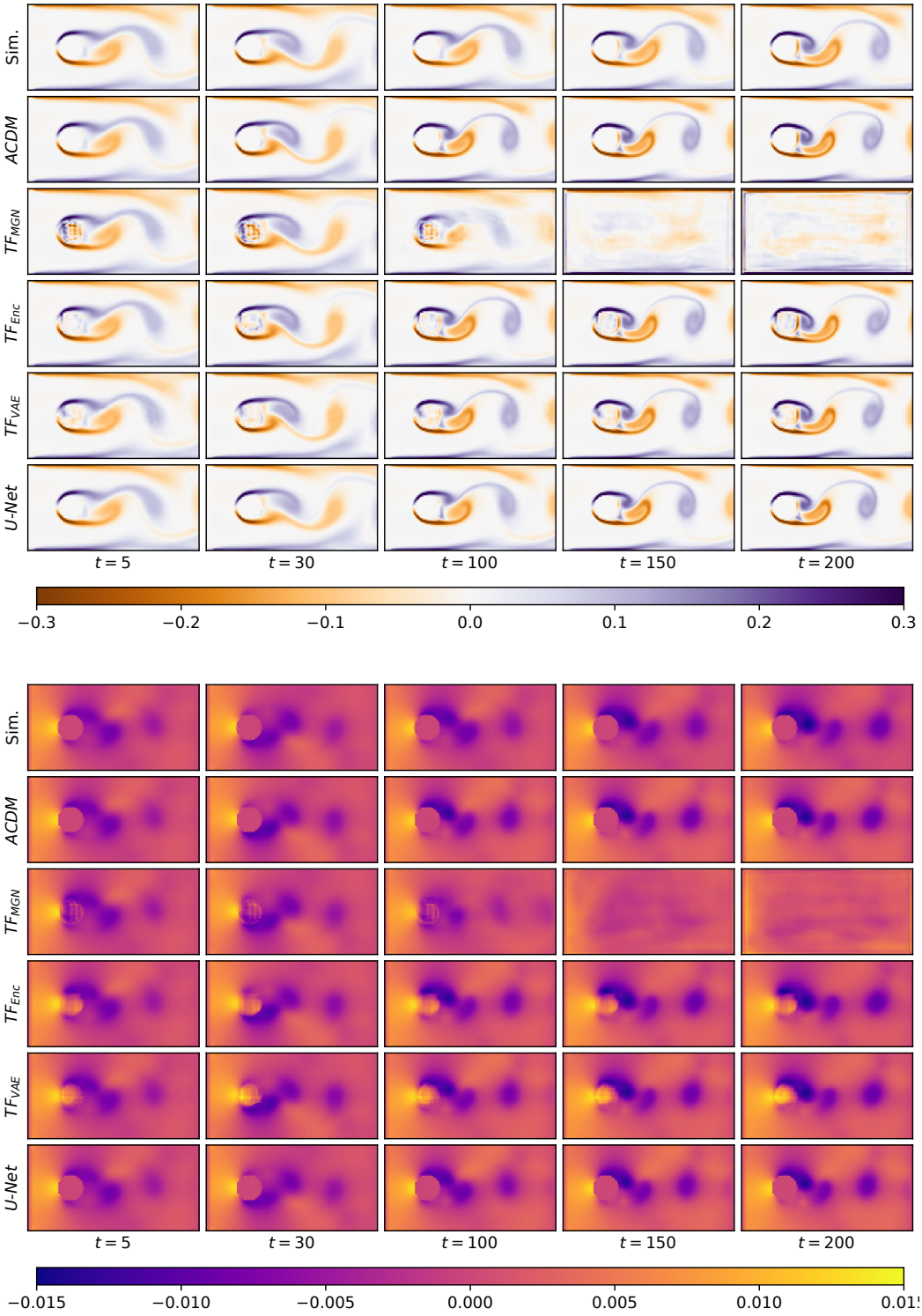

Figure 15: Predictions of various models for the $\texttt{Inc}_{\texttt{var}}$ sequence. Shown are vorticity at the top and pressure at the bottom.

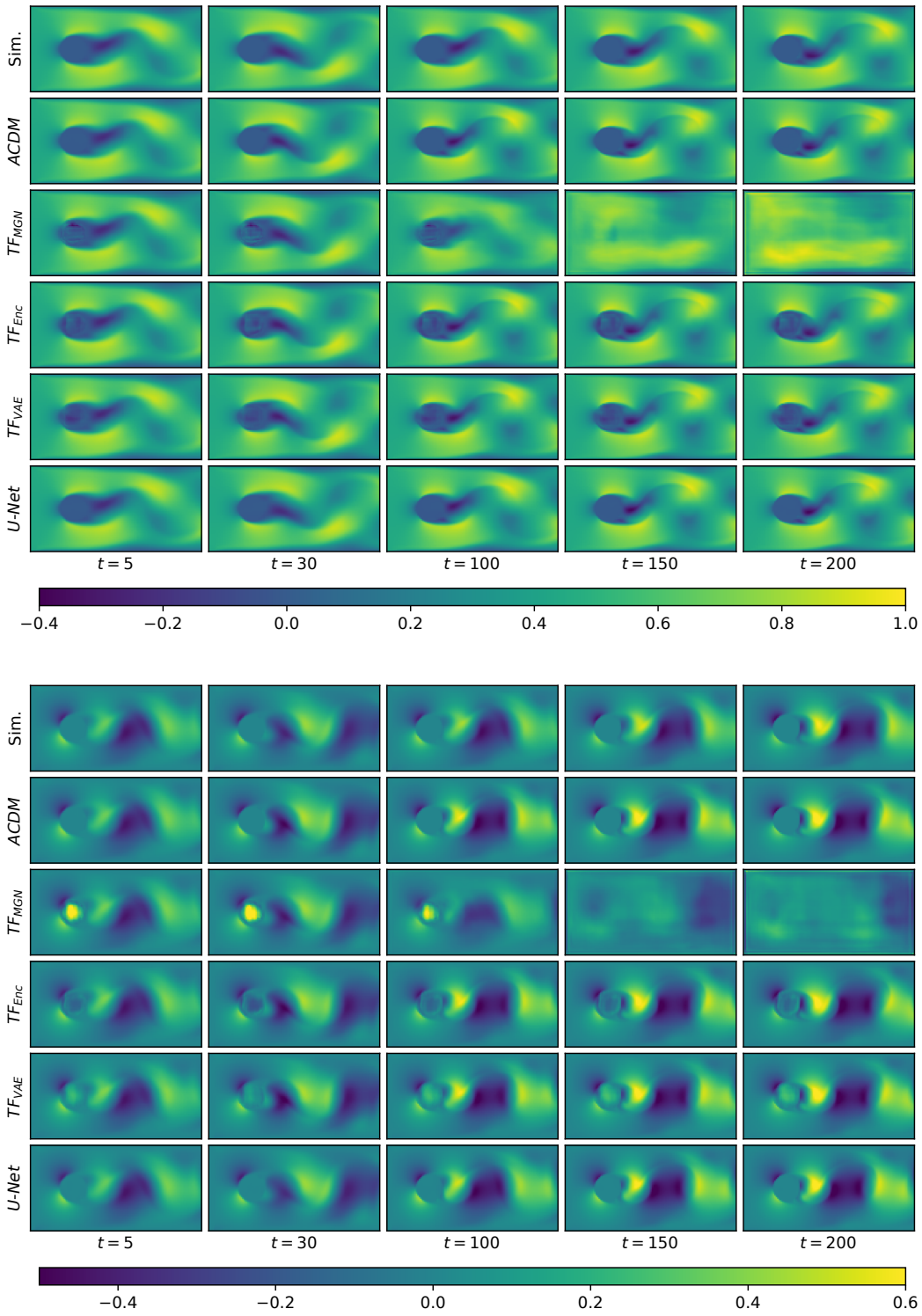

Figure 16: Predictions of various models for the $\texttt{Inc}_{\texttt{var}}$ sequence. Shown are the velocity x-component at the top and the y-component at the bottom.

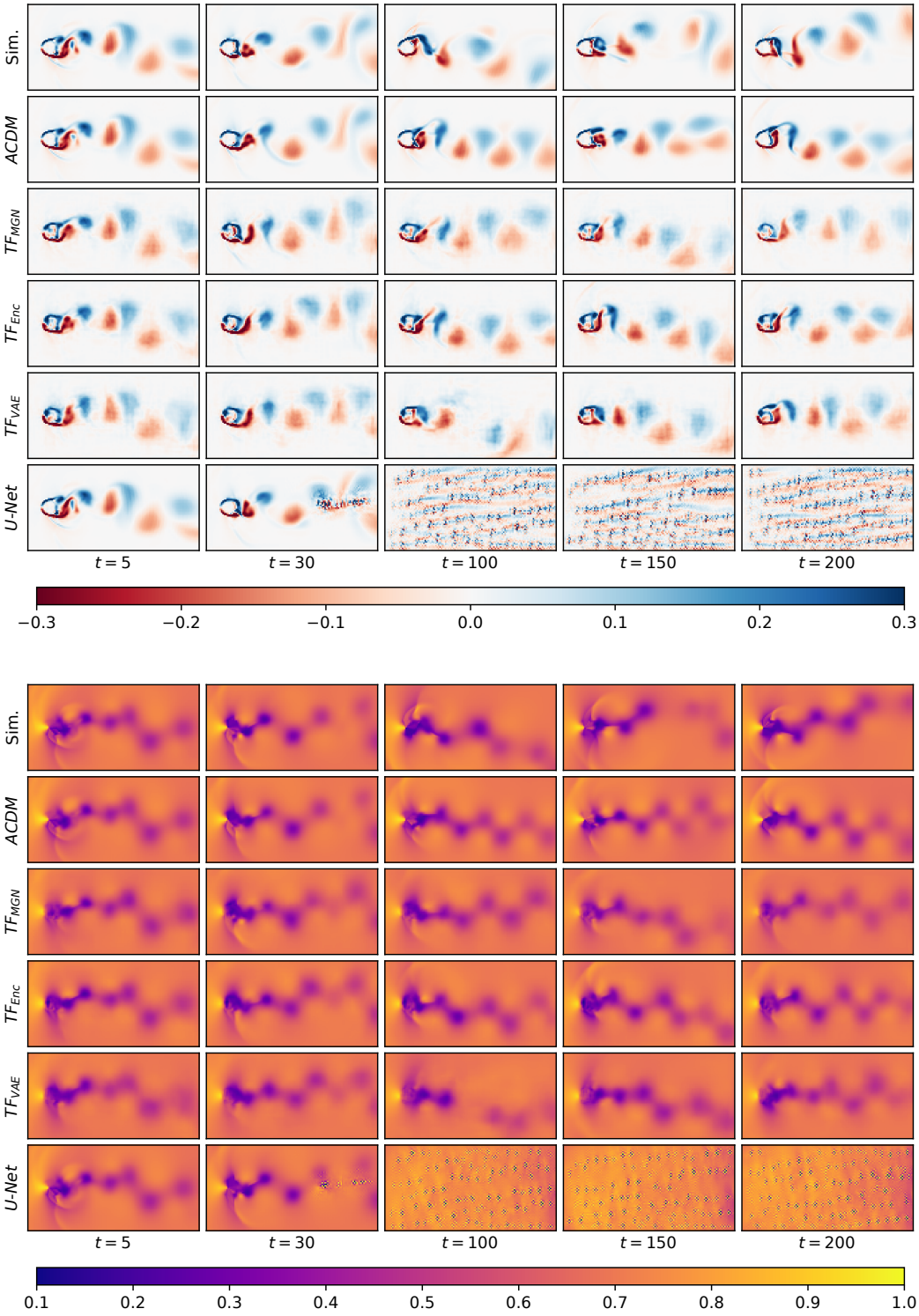

Figure 17: Predictions of various models for example sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$. Shown are vorticity at the top and pressure at the bottom.

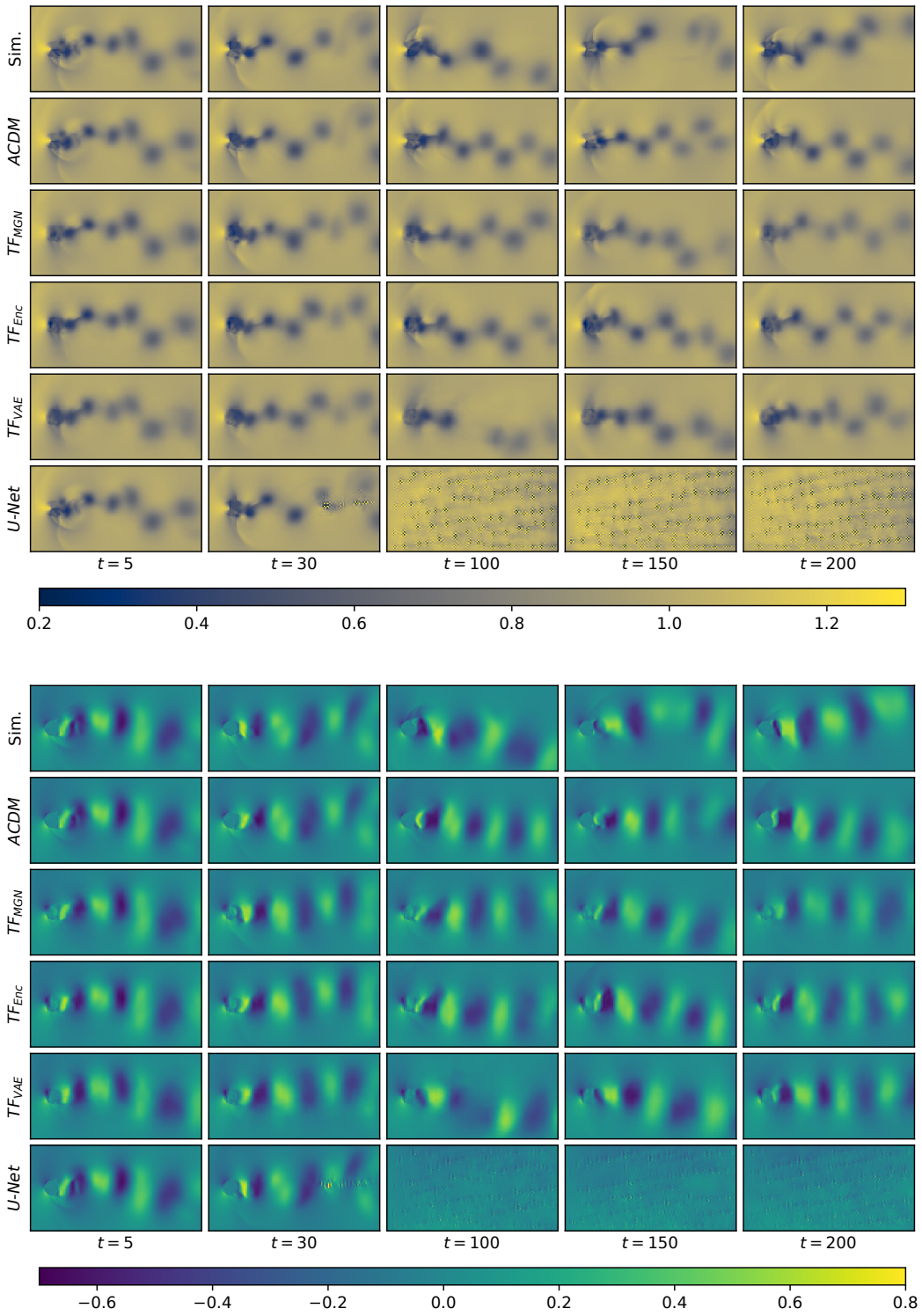

Figure 18: Predictions of various models for example sequence from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$. Shown are density at the top and the velocity y-component at the bottom.

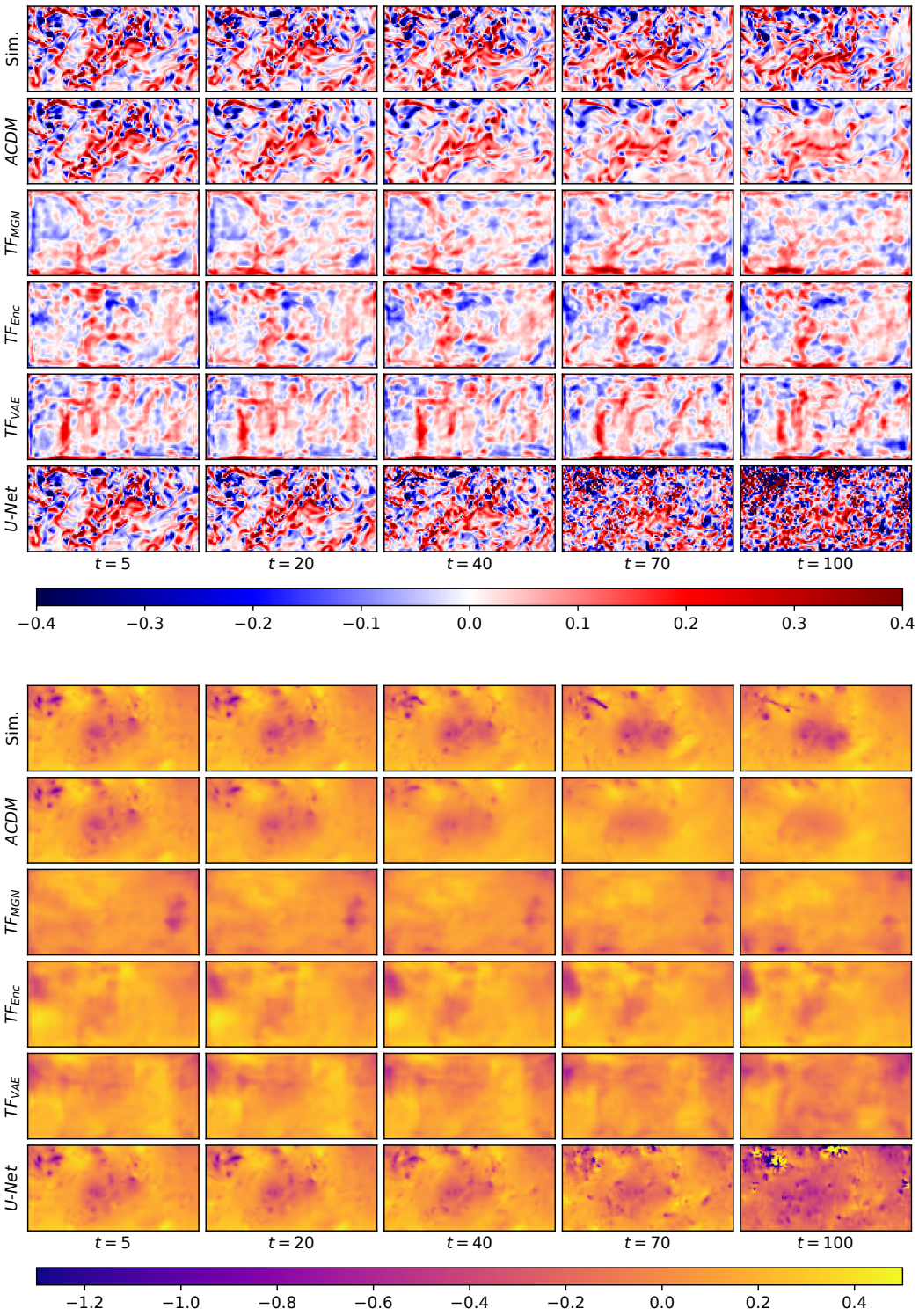

Figure 19: Predictions of various models for example sequence from `Iso` with $z = 280$. Shown are vorticity at the top and pressure at the bottom.

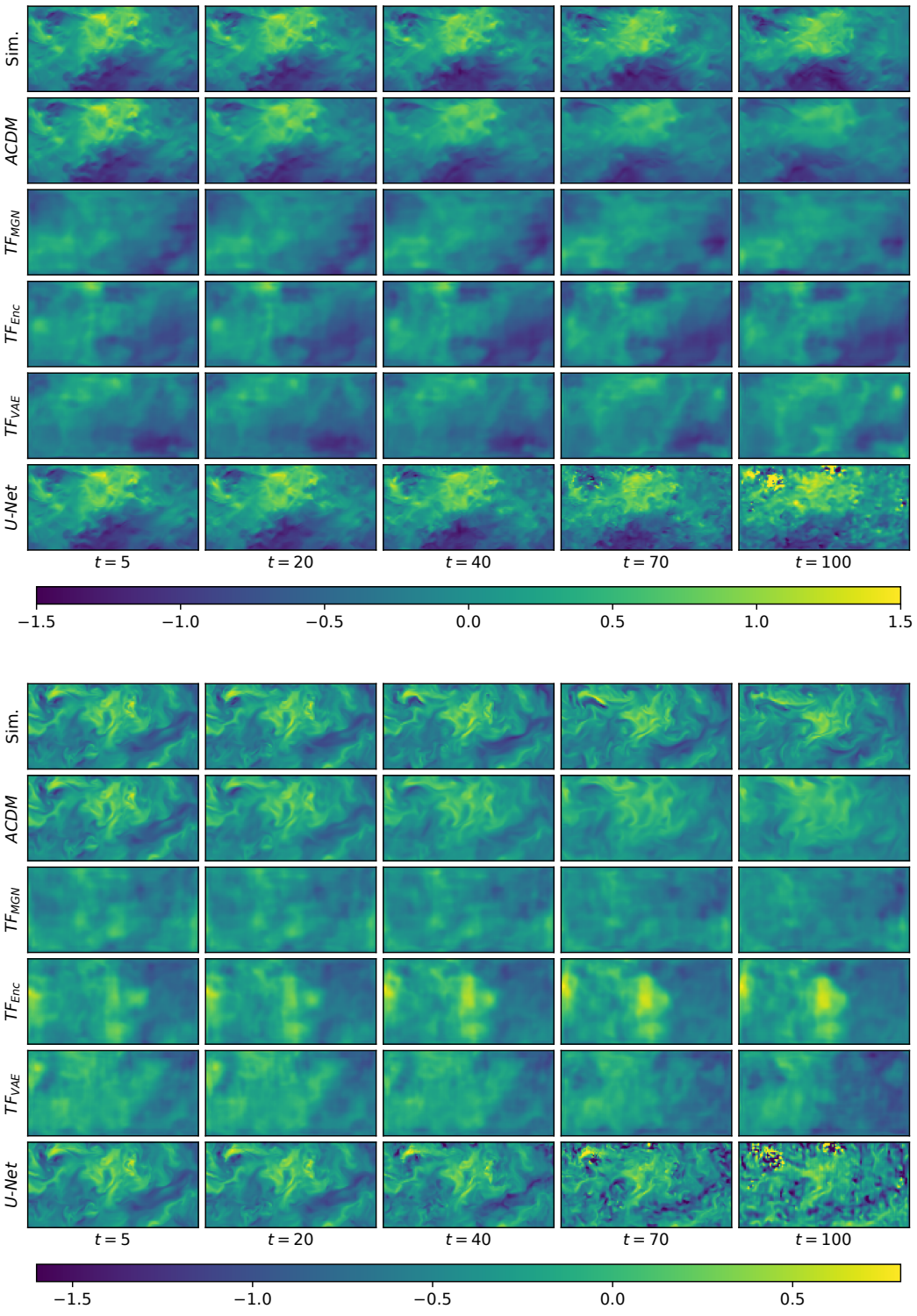

Figure 20: Predictions of various models for example sequence from `Iso` with $z = 280$. Shown are the velocity x-component at the top and the z-component at the bottom.

C.3 Additional Variational Autoencoder Evaluations

Temporal Coherence Here, we analyze the temporal coherence between individual time steps of the TF_{VAE} model. As an important difference to *ACDM*, the decoder of TF_{VAE} does not have access to previously generated time steps, as its 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. In Fig. 21 on the left, we display the first three simulation steps of a sequence from `Iso`, along with the corresponding predictions of TF_{Enc} and TF_{VAE}. In addition, the change between the first two predicted steps and s^0 is shown on the right. While both methods struggle to reproduce the original vorticity field at $t = 0$, there is a clear difference between both trajectories: the distance between the predictions at $t = 0$ and $t = 1$ is relatively small for TF_{Enc}, but big for TF_{VAE}. This results in an even visually noticeable jump in the predictions of TF_{VAE} (also see accompanying videos).

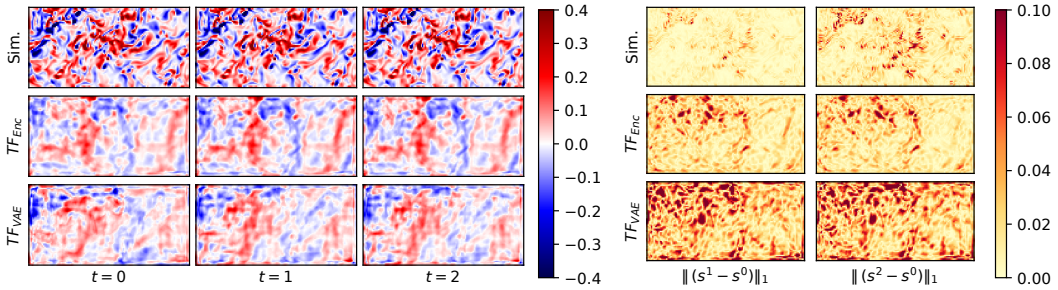

Figure 21: Temporal coherence of TF_{Enc} and TF_{VAE} (left) and comparison difference between reconstruction and first two prediction steps for both models (right) on example from `Iso` with $z = 300$.

Posterior Sampling Similar to the posterior sampling evaluation for *ACDM*, we also analyze the posterior samples create by the TF_{VAE} model. We use the same sequence, time step, and zoomed sample area as shown in the main paper. Figure 22 contains three random TF_{VAE} example samples and a spatial standard deviation for all five samples across different time steps on $\texttt{Tra}_{\texttt{long}}$ and `Iso`. Note that the zoomed samples are displayed via Catmull-Rom spline interpolation for visual clarity in this visualization. Compared to *ACDM*, all samples are generally highly similar and exhibit very little variance across random model evaluations. While small scale details are varying, the overall structure, e.g., vortex positions for $\texttt{Tra}_{\texttt{long}}$ or areas of high vorticity for `Iso`, is identical for each sample. Furthermore, the variance does not substantially increase over time as it would be expected. Due to the inherent data compression, TF_{VAE} introduces some artifacts which are especially noticeable in the vortex shapes for $\texttt{Tra}_{\texttt{long}}$ on the top left in Fig. 22. Furthermore, it also struggles to create fine details like the shock waves in $\texttt{Tra}_{\texttt{long}}$ as shown at the bottom left.

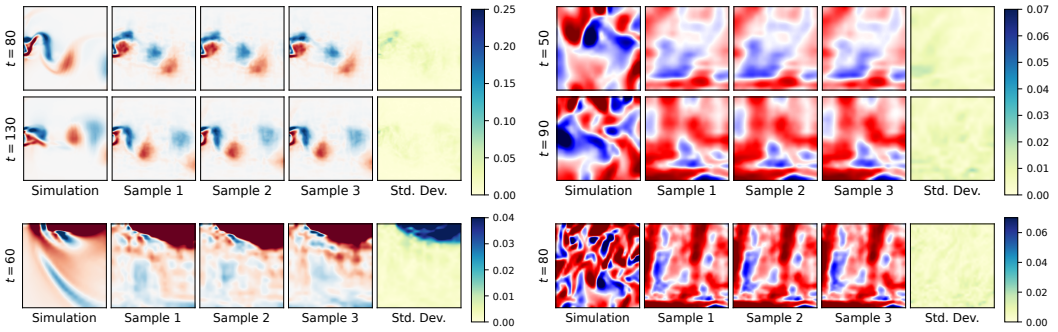

Figure 22: Zoomed TF_{VAE} posterior samples with corresponding standard deviation from $\texttt{Tra}_{\texttt{long}}$ with $Ma = 0.64$ (left), and from `Iso` with $z = 300$ (right) at different time steps t.

C.4 Accuracy Ablations

Table 1: Quantitative comparison across our test sets for different network architectures (best and second best results are highlighted for each data 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})
TF_{MGN}	5.9±4.5	14±6.1	18±10	4.5±1.2	3.5±0.9	1.8±0.3	6.1±4.2	2.1±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	7.1±2.1
TF_{VAE}	5.5±5.6	13±7.3	11±6.3	3.8±0.9	4.0±0.9	2.3±0.2	7.2±3.1	2.7±0.6	10±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
ACDM (ours)	1.4±1.9	6.3±5.5	0.9±0.5	0.9±0.2	2.5±1.5	1.3±0.3	2.7±2.5	1.3±0.6	3.4±0.9	3.6±0.6

Table 1 contains the full, numerical accuracy values corresponding to Fig. 4. Shown are the mean-squared-error (MSE) and LSiM, a similarity metric for data from numerical simulations designed to more accurately capture larger patterns or connected structures (Kohl et al., 2020). For both metrics, lower values indicate better reconstruction accuracy, and rollout errors reported, i.e., computed per time step and field, and averaged over the full temporal rollout.

In the following, we provide ablation studies for model parameters of *U-Net* and *ACDM* in terms of accuracy. First, we investigate the impact of the training rollout length m for *U-Net*. For these models, we use the same U-Net architecture as described in Appendix B.2 with $k = 1$, but propagate gradients through multiple state predictions during training. The bottom of Tab. 2 shows the accuracy of models trained with different rollout lengths on the test sets $\texttt{Tra}_{\texttt{ext}}$ and $\texttt{Tra}_{\texttt{int}}$, while the top of the table contains the original *ACDM* and *U-Net* for reference. For the transonic flow, already $m = 4$ is sufficient to substantially improve the training 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 in Appendix C.5.

Table 2: Ablations for training rollout length m on $\texttt{Tra}_{\texttt{ext}}$ and $\texttt{Tra}_{\texttt{int}}$.

	$\texttt{Tra}_{\texttt{ext}}$		$\texttt{Tra}_{\texttt{int}}$	
Method	MSE (10^{-3})	LSiM (10^{-1})	MSE (10^{-3})	LSiM (10^{-1})
ACDM	2.2±1.3	1.3±0.2	2.7±2.5	1.3±0.6
U-Net	3.1±2.1	3.9±2.8	2.3±2.0	3.3±2.8
$U\text{-}Net_{m4}$	1.6±1.0	1.4±0.8	1.1±1.0	0.9±0.4
$U\text{-}Net_{m8}$	1.6±0.7	1.1±0.2	1.5±1.5	1.0±0.5
$U\text{-}Net_{m16}$	2.2±1.1	1.3±0.3	2.4±1.3	1.3±0.5

On Iso in Tab. 3, the behavior 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*, e.g., denoted by *$U\text{-}Net_{m4,Pre}$*. With this pre-training the previous behavior emerges, and *$U\text{-}Net_{m8,Pre}$* even clearly improves upon *$U\text{-}Net_{m4}$*. Compared to *ACDM*, the variants of *U-Net* with longer rollouts can achieve similar or slightly higher accuracy by significantly reducing the complexity of the training task, and thus the models ability to generalize. Pre-training with shorter training rollouts is also beneficial.

Table 3: Ablations for training rollout length m and pre-training on Iso.

	Iso	
Method	MSE (10^{-2})	LSiM (10^{-1})
ACDM	3.4±0.9	3.6±0.6
U-Net	25.8±35	11.3±3.9
$U\text{-}Net_{m4}$	3.7±0.8	2.8±0.5
$U\text{-}Net_{m8}$	4.5±2.8	2.4±0.5
$U\text{-}Net_{m16}$	13.0±11	3.8±1.5
$U\text{-}Net_{m4,Pre}$	5.7±2.6	3.6±0.8
$U\text{-}Net_{m8,Pre}$	2.6±0.6	2.3±0.5
$U\text{-}Net_{m16,Pre}$	2.9±1.4	2.3±0.5

C.5 Additional Temporal Stability Evaluations

Below, we evaluate the temporal stability properties of the *U-Net* models with longer training rollouts. As detailed in the main paper, we measure the magnitude of the rate of change of s for this purpose. This metric remains reliable on cases with complex data and long inference rollouts, and illustrates if

models follow the state evolution behavior given by the reference simulation. The top of Fig. 23 shows the temporal stability on $\texttt{Tra}_{\texttt{long}}$ for *U-Net*$_{m4}$, *U-Net*$_{m8}$, and *U-Net*$_{m16}$. *ACDM* and *U-Net* are also included for reference. Evaluated are all trained models for every sequence from the corresponding test set, and the shaded area corresponds to the standard deviation. All models perform similar until about $t = 50$ where *U-Net* deteriorates. *U-Net*$_{m4}$ also exhibits similar signs of deterioration around $t = 130$ during the rollout. Only *U-Net*$_{m8}$ and *U-Net*$_{m16}$ are fully stable across the entire rollout of $T = 240$ steps and performs comparable to *ACDM*.

The bottom left of Fig. 23 contains evaluations on $\texttt{Iso}$ for the same *U-Net* variants described above. *U-Net*$_{m4}$ and *U-Net*$_{m8}$ achieve comparable stability to *ACDM* in this case, with an almost constant rate of change for the entire rollout. Models with shorter rollouts, i.e., *U-Net* deteriorate after an initial phase, and longer rollouts prevent effective training for *U-Net*$_{m16}$ as explained in Appendix C.4. Variations with additional pre-training are shown on the bottom right in Fig. 23. *U-Net*$_{m4,Pre}$ does not substantially improve upon *U-Net*$_{m4}$, and *U-Net*$_{m8,Pre}$ performs similar to *U-Net*$_{m8}$ and very well. Only for the longer rollouts in the *U-Net*$_{m16,Pre}$ model pre-training helps, such that *U-Net*$_{m16,Pre}$ is fully stable and in line with the ideal rollout length of $m = 8$.

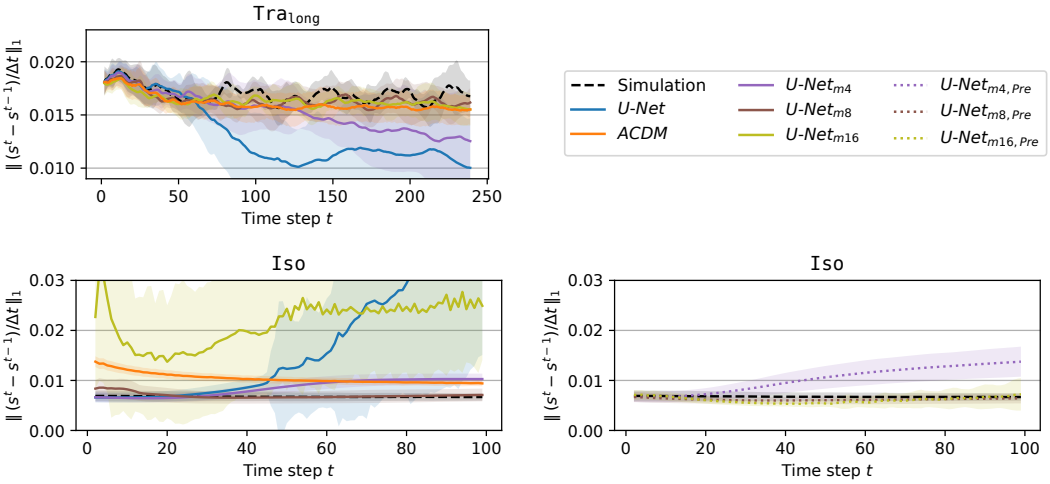

Figure 23: Stability analysis via difference to previous time step on $\texttt{Tra}_{\texttt{long}}$ (top) and $\texttt{Iso}$ (bottom).