A Appendix

B Diffusion process as ODE

In this section, we treat $\mathbf{x}$ as a continuous function of time, i.e. let $\mathbf{x} : \mathbb{R} \to \mathbb{R}^n$, $s \mapsto \mathbf{x}(s)$. Here s denotes the time variable (since t is already taken and n usually refers to discrete variables).

During prediction, $\mathbf{x}_t$ is given, ϕ is fixed, and $\mathcal{I}_\phi$ only depends on $\hat{\mathbf{x}}_{t+h}$ and i. Therefore, we simplify notations by writing $\mathcal{I}_\phi\left(\mathbf{x}_t, \hat{\mathbf{x}}_{t+h}, i\right) = \mathcal{I}_{\phi,\mathbf{x}_t}\left(\hat{\mathbf{x}}_{t+h}, i\right)$. We will further omit the subscripts ϕ and $\mathbf{x}_t$.

B.1 Cold Sampling from the Euler method

In this section, we show that Cold Sampling is an approximation of the Euler method for (5).

The Euler method for integrating $\mathbf{x}$ is

$$\mathbf{x}_{s+\Delta s} = \mathbf{x}_s + \Delta s \frac{d\mathcal{I}(F_\theta(\mathbf{x}_s, s), s)}{ds} \tag{7}$$

for a small Δs. We do not have access to $\frac{d\mathcal{I}(F_\theta(\mathbf{x}_s,s),s)}{ds}$. However, since we know F_θ and $\mathcal{I}_\phi$, we can approximate $\frac{d\mathcal{I}(F_\theta(\mathbf{x}_s,s),s)}{ds}$ by its first-order Taylor expansion around s:

$$\Delta s \frac{d\mathcal{I}(F_\theta(\mathbf{x}_s, s), s)}{ds} \approx \mathcal{I}(F_\theta(\mathbf{x}_{s+\Delta s}, s+\Delta s), s+\Delta s) - \mathcal{I}(F_\theta(\mathbf{x}_s, s), s) \tag{8}$$

This step can also be interpreted as evaluating the integral in (6) using the fundamental theorem of calculus. Then the Euler method becomes

$$\mathbf{x}_{s+\Delta s} = \mathbf{x}_s + \mathcal{I}(F_\theta(\mathbf{x}_{s+\Delta s}, s+\Delta s), s+\Delta s) - \mathcal{I}(F_\theta(\mathbf{x}_s, s), s) \tag{9}$$

Note that $\mathbf{x}_{s+\Delta s}$ on the right hand side is unknown, because it is the quantity we want to approximate in this step. A reasonable way to approximate $F_\theta(\mathbf{x}_{s+\Delta s}, s+\Delta s)$ is to replace it by $F_\theta(\mathbf{x}_s, s)$, because they both predict $\mathbf{x}(h)$ and use nearby points (assuming Δs is small and $\mathbf{x}$ behaves nicely around s). The resulting update,

$$\mathbf{x}_{s+\Delta s} = \mathbf{x}_s + \mathcal{I}(F_\theta(\mathbf{x}_s, s), s+\Delta s) - \mathcal{I}(F_\theta(\mathbf{x}_s, s), s), \tag{10}$$

is exactly the Cold Sampling algorithm (Alg. 2). By formulating the diffusion process as an ODE, we have provided a new theoretical explanation for Cold Sampling.

Note that we made the approximation that $F_\theta(\mathbf{x}_{s+\Delta s}, s+\Delta s) \approx F_\theta(\mathbf{x}_s, s)$ to obtain the update rule (previous equation). The error introduced by this approximation is expected to be larger when s is small. The intuition is as follows. When s is small, the distance between s and $t+h$ is large, and $F_\theta(\mathbf{x}_s, s)$ has to make a prediction farther ahead. Predicting far into the further is generally harder than predicting the near future. Therefore, the prediction error $F_\theta(\mathbf{x}_s, s) - \mathbf{x}_{t+h}$ is expected to be larger when s is small. The larger uncertainty may lead to larger difference between $F_\theta(\mathbf{x}_{s+\Delta s}, s+\Delta s)$ and $F_\theta(\mathbf{x}_s, s)$.

To reduce the error brought by this approximation, it makes sense to sample more densely around the early part of the prediction window.

B.2 Why is cold sampling better than naive sampling?

In experiments, and consistent with prior work [2], cold sampling outperforms naive sampling by a large margin. We provide an explanation by analyzing the discretization errors in the two sampling algorithms. In cold sampling, the discretization error per step is bounded by a term proportional to the step size Δs. Naive sampling does not have this property.

The true value of $\mathbf{x}$ at $s + \Delta s$ according to Equation (6) is

$$\begin{aligned}\mathbf{x}(s+\Delta s) &= \mathbf{x}(s) + \int_s^{s+\Delta s} \frac{d\mathcal{I}_\phi(F_\theta(\mathbf{x}, s), s)}{ds} \\ &= \mathbf{x}(s) + \mathcal{I}(F_\theta(\mathbf{x}(s+\Delta s), s+\Delta s), s+\Delta s) - \mathcal{I}(F_\theta(\mathbf{x}(s), s), s).\end{aligned} \tag{11}$$

Recall from Alg. 2 that given $\mathbf{x}(s)$, cold sampling predicts $\mathbf{x}(s+\Delta s)$ as

$$\hat{\mathbf{x}}(s+\Delta s) = \mathbf{x}(s) + \mathcal{I}_\phi(\mathbf{x}_{t-l:t}, F_\theta(\mathbf{x}(s), s), s+\Delta s) - \mathcal{I}_\phi(\mathbf{x}_{t-l:t}, F_\theta(\mathbf{x}(s), s), s). \tag{12}$$

The discretization error $e(\mathbf{x})$ of one step of cold sampling is the difference between the exact and predicted $\mathbf{x}(s+\Delta s)$:

$$\begin{aligned} e(\mathbf{x}, \Delta s) &= \mathbf{x}(s+\Delta s) - \hat{\mathbf{x}}(s+\Delta s) \\ &= \mathcal{I}(F_\theta(\mathbf{x}(s+\Delta s), s+\Delta s), s+\Delta s) - \mathcal{I}(F_\theta(\mathbf{x}(s), s), s+\Delta s). \end{aligned} \tag{13}$$

The following proposition states that $e(\mathbf{x})$ is bounded by a term proportional to the step size Δs.

Proposition B.1. *Assume that $F_\theta(\mathbf{x}(s), s)$ is Lipschitz in s. Assume also that $\mathcal{I}_\phi(\mathbf{x}_{t+h}, s)$ is Lipschitz in $\mathbf{x}_{t+h}$. The norm of the cold sampling discretization error, $||e(\mathbf{x}, \Delta s)||_2$, is bounded by $O(\Delta s)$.*

Proof. The proof relied on applying definitions of Lipschitz functions twice. Let L_1 be the Lipschitz constant for $F_\theta(\mathbf{x}(s), s)$ in s. Let L_2 be the Lipschitz constant for $\mathcal{I}(\mathbf{x}, s)$ in $\mathbf{x}$. Since $F_\theta(\mathbf{x}(s), s)$ is Lipschitz in s, we have $||F_\theta(\mathbf{x}(s+\Delta s), s+\Delta s) - F_\theta(\mathbf{x}(s), s)||_2 \leq L_1 \Delta s$. Since $\mathcal{I}(\mathbf{x}, s)$ is Lipschitz in $\mathbf{x}$, we have $||\mathcal{I}(F_\theta(\mathbf{x}(s+\Delta s), s+\Delta s), s+\Delta s) - \mathcal{I}(F_\theta(\mathbf{x}(s), s), s+\Delta s)||_2 \leq L_2 L_1 \Delta s$. Therefore $||e(\mathbf{x})|| \leq L_2 L_1 \Delta s$, which means the discretization error is bounded by a first-order term of the step size. □

Under the same Lipschitz assumptions, the discretization error of the naive sampling is not guaranteed to be in the first order of step size. In naive sampling, the predicted $\mathbf{x}$ at time $s+\Delta s$ is

$$\hat{\mathbf{x}}(s+\Delta s) = \mathcal{I}(F_\theta(\mathbf{x}(s), s), s+\Delta s). \tag{14}$$

The discretization error of one step of naive sampling is $\mathbf{x}(s+\Delta s)$:

$$\begin{aligned} e(\mathbf{x}, \Delta s) =& \mathbf{x}(s+\Delta s) - \hat{\mathbf{x}}(s+\Delta s) \\ =& \mathcal{I}(F_\theta(\mathbf{x}(s+\Delta s), s+\Delta s), s+\Delta s) - \mathcal{I}(F_\theta(\mathbf{x}(s), s), s+\Delta s) \\ &+ \mathbf{x}(s) - \mathcal{I}(F_\theta(\mathbf{x}(s), s), s). \end{aligned} \tag{15}$$

Note that the first two terms are the same as the discretization error in cold sampling. However, the last two term are not bounded by first-order terms of Δs. Hence, naive sampling can have larger discretization errors.

B.3 Evaluation

We use the implementation in the `xskillscore`[3] Python package to compute the CRPS of the ensemble forecasts.

B.4 Datasets

B.4.1 SST Data Preprocessing

We create a new *sea surface temperatures* (SST) dataset based on NOAA OISSTv2 [30], which comes at a daily time-scale. These data is available from 1982 to the present at a resolution of $1/4°$ degrees. For training we use the years 1982-2018, for validation 2019, and for testing 2020. We have preprocessed the NOAA OI SST V2 dataset as follows:

1. First, the globe is divided into 60×60 latitude $\times$ longitude grid tiles,
2. all tiles with less than 95% of ocean cover are filtered out,
3. standardize the raw SSTs using daily means and standard deviations (computed on the training set only, i.e. 1982-2018),

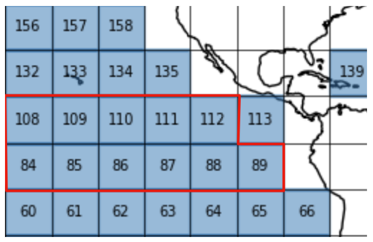


Figure 4: Visualization of the SST dataset that we created. It divides the globe into 60×60 latitude $\times$ longitude grid tiles. We only use the subset delineated in red, i.e. boxes 84-89 and 108-112.

[3] `https://xskillscore.readthedocs.io/`

Table 4: The hyperparameters used for each dataset. For the learning rates, we sweep over each value and report the best set of runs based on their validation CRPS computed on 50 samples. For architectural details, see B.5.2.

Hyperparameters for each dataset

Hyperparameter	SST	Navier-Stokes	Spring Mesh
Batch size	64	32	64
Accumulate gradient batches	4	2	1
Max. Epochs	50	200	300
Gradient clipping (norm)	1.0	1.0	1.0
Learning rate(s)	$7e$-4, $3e$-4, $5e$-5, $1e$-5	$7e$-4, $3e$-4	$4e$-4
Weight decay	$1e$-5	$1e$-4	$1e$-4
AdamW β_1	0.9	0.9	0.9
AdamW β_2	0.99	0.99	0.99

4. replace continental NaNs with zeroes (after standardization), and

5. we subsample 11 grid tiles (covering mostly the eastern tropical Pacific, as shown in Fig. 4).

B.5 Experiments

B.5.1 Implementation Details

The set of hyperparameters that we use for each dataset, such as the learning rate and maximum number of epochs, can be found in Table 4. For all experiments we use a floating point precision of 16, and do not use a learning rate scheduler. All diffusion models, including DYffusion, are trained with the L1 loss, while all bare-bone UNet/CNN networks are trained on the L2 loss. We use three different dropout rates for the SST UNet: 1) before the query-key-value projection of each attention layer, dr_{at}; 2) After the first sub-block of each ResNet block, dr_{bl_1}; 3) After the second sub-block of each ResNet block, dr_{bl_2}, where the first ResNet sub-block consists of convolution $\rightarrow$ normalization $\rightarrow$ time-embedding scale-shift $\rightarrow$ activation function, and the second sub-block is the same but without the time-embedding scale-shift.

Perturbation baseline We perturb the initial conditions, $\mathbf{x}_t$, with small amounts of Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon \mathbf{I})$. We found that $\sigma_\epsilon^* = 0.05$ gave the lowest CRPS scores among all variances that we tried, $\sigma_\epsilon \in \{0.01, 0.03, 0.05, 0.07, 0.1, 0.15, 0.2\}$. We note that choosing larger variances results in better SSR scores, but significantly lower CRPS and MSE scores. Inference dropout was disabled for this baseline variant.

Dropout baseline For this baseline, we enable the bare-bone model's dropout during both training and inference. For the SST dataset, similarly to the interpolator network of DYffusion, we found that using high dropout rates results in better performance. An explanation could be that the SST UNet has capacity than the other backbone architectures. Concretely, the following SST UNet dropout rates resulted in best performance: $dr_{at} = dr_{bl_2} = 0.6, dr_{bl_1} = 0.3$. For Navier-Stokes and spring mesh there is only one dropout hyperparameter, and the corresponding best model uses 0.2 and 0.05 as dropout rate, respectively (selected from a sweep over $\{0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$).

DDPM We found that the cosine (linear) noise schedule gives better results for the SST (Navier-Stokes) dataset, and always use the "predict noise" objective. For the SST dataset, the best performing DDPM is trained with 5 diffusion steps, while for Navier-Stokes it is trained with 500 steps. For Navier-Stokes we found that while a DDPM with 5 or 10 diffusion steps can give good validation scores (or even better ones than the 500-steps DDPM), it ends up diverging at test time after a few autoregressive iterations when used to forecast full trajectories.

MCVD [65] We train MCVD with 1000 diffusion steps for all datasets, as we were not able to successfully train it with fewer diffusion steps. We use a linear noise schedule (we found the cosine schedule to produce inferior results) using the "predict noise" objective. Due to the inference runtime complexity of using 1000 diffusion steps, we only report one MCVD run in our main SST results.

DYffusion For the SST dataset we use 35 artificial diffusion steps (analogous to the schedule in green in Fig. 2), while for the Navier-Stokes and spring mesh datasets we do not use any, i.e. $S = [j]_{j=0}^{h-1}$. Furthermore, we found that the refinement step of Alg. 2 did not improve performance for the SST dataset so we did not use it there, whereas it did improve performance for Navier-Stokes and spring mesh. As for the choice of the interpolator network, $\mathcal{I}_\phi$, we conduct a sweep over the dropout rates for each dataset (analogous to the "Dropout" baseline). This is an important hyperparameter, since we found that stochasticity in the interpolator is crucial for the overall performance of DYffusion. The interpolator network is selected based on the lowest validation CRPS. For the SST dataset, the selected $\mathcal{I}_\phi$ uses $dr_{at} = dr_{bl_2} = 0.6, dr_{bl_1} = 0$. The dropout rates for Navier-Stokes and spring mesh are 0.15 and 0.05, respectively. Generally, we found that the optimal amount of dropout for any given dataset strongly correlates between the "Dropout" multi-step forecasting baseline and DYffusion's interpolator network, $\mathcal{I}_\phi$. Thus, for a new dataset or problem, it is a valid strategy to sweep over the dropout rate for just one of the two model types. Motivated by the intuition that temporal interpolation is a simpler task than forecasting, the channel dimensionality of $\mathcal{I}_\phi$ is only 32 (instead of 64) for the first downsampling block of the SST UNet.

B.5.2 Neural architecture details

SST UNet For the SST dataset, we use a UNet implementation commonly used as backbone architecture of diffusion models[4]. The UNet for the SST dataset consists of three downsampling and three upsampling blocks. Each blocks consists of two convolutional residual blocks (ResNet blocks), followed by an attention layer, and a downsampling (or upsampling) module. Each ResNet block can be further divided into two sub-blocks. The first one consists of convolution → normalization → time-embedding scale-shift → activation function, and the second sub-block is the same but without the time-embedding scale-shift. The downsampling module is a 2D convolution that halves the spatial size of the input (with a 4×4 kernel and stride$= 2$). The upsampling module doubles the spatial size via nearest neighbor upsampling followed by a 2D convolution (with 3×3 kernel). At the end of each downsampling (upsampling) block the spatial size is halved (doubled) and the channel dimension is doubled (halved). We use 64 channels for the initial downsampling block, which means that the channel dimensionalities are $64 \rightarrow 128 \rightarrow 256$ and the spatial dimensions $(60, 60) \rightarrow (30, 30) \rightarrow (15, 15)$ in the corresponding downsampling blocks (reversed for the upsampling blocks). We use three different dropout rates for the SST UNet: 1) before the query-key-value projection of each attention layer, dr_{at}; 2) After the first sub-block of each ResNet block, dr_{bl_1}; 3) After the second sub-block of each ResNet block, dr_{bl_2}. For all models except the time-conditioned bare-bone network and $\mathcal{I}_\phi$ in DYffusion, which use higher dropout rates, we use $dr_{at} = 0.1, dr_{bl_1} = 0, dr_{bl_2} = 0.3$.

Navier-Stokes UNet and spring mesh CNN For the Navier-Stokes and spring mesh benchmark datasets from [43], we simply re-use their proposed UNet and CNN architecture for the respective dataset. The only change is, that we integrate the same time embedding module from the SST UNet, as described below.

Time embedding module The time-embedding scale-shift, taken from the SST UNet, is a key component of all architectures since it enables them to condition on the diffusion step (for DDPM and MCVD) or the dynamical timestep (for the time-conditioned bare-bone models as well as for both F_θ and $\mathcal{I}_\phi$ in DYffusion). It is implemented by a sine-cosine based featurization of the scalar diffusion step/dynamical timestep. These features are projected by a linear layer, followed by a GeLU activation, and another linear layer, which results in a "time embedding". Then, separately for each convolutional (or ResNet) block of the neural architecture the "time embedding" is further processed by a SiLU activation and another linear layer whose output is interpreted as two vectors which are used to scale and shift the block's inputs. In all architectures, the scale-shift operation is performed after convolution and normalization layers, but before the activation function and dropout layer.

Table 5: Sampling ablation. We change one component at a time in DYffusion, starting with all components enabled (first row). For SST, we perform the ablation only on a subset of the test dataset (box 88 in Fig. 4). For the Navier-Stokes and spring mesh datasets, we use the full test sets. *No ref.* refers to not using the refinement step in line 6 of Alg. 2. *No dr.* refers to disabling the inference dropout of the interpolator network, $\mathcal{I}_\phi$. No Dr.& σ_ϵ refers to disabling the inference dropout of $\mathcal{I}_\phi$ and perturbing the inputs by $\sigma_\epsilon = 0.05$ (like for the Perturbation baseline).

Change	**SST**			**Navier-Stokes**			**Spring Mesh**		
	CRPS	MSE	SSR	CRPS	MSE	SSR	CRPS	MSE	SSR
Full	0.182 ± 0.001	0.111 ± 0.001	1.03 ± 0.02	0.067 ± 0.003	0.022 ± 0.002	0.88 ± 0.01	0.0107 ± 0.0025	4.74e-04 $\pm$ 2.38e-04	1.11 ± 0.09
No ref.	0.182 ± 0.001	0.111 ± 0.001	1.08 ± 0.00	0.069 ± 0.003	0.024 ± 0.002	1.12 ± 0.02	0.0249 ± 0.0014	7.62e-04 $\pm$ 3.10e-04	2.02 ± 0.15
No Dr.	0.320 ± 0.009	0.206 ± 0.012	0.00 ± 0.00	0.098 ± 0.005	0.028 ± 0.003	0.00 ± 0.00	0.0348 ± 0.0042	2.77e-03 $\pm$ 6.71e-04	0.00 ± 0.00
No Dr.& σ_ϵ	0.308 ± 0.009	0.197 ± 0.012	0.40 ± 0.01	0.070 ± 0.004	0.024 ± 0.002	0.85 ± 0.03	0.0292 ± 0.0034	2.98e-03 $\pm$ 6.79e-04	0.96 ± 0.05

B.5.3 Ablations

Inference dropout in the interpolator net In Table 5 we show that disabling the inference dropout in the interpolator network, $\mathcal{I}_\phi$, results in considerably worse scores. This is to be expected, since without stochasticity in $\mathcal{I}_\phi$ our current framework collapses to forecasting deterministically (since the sampling algorithm and forecaster network are deterministic, and we assume that the given initial conditions are fixed). In such a case, computing the CRPS collapses to the mean absolute error, and the SSR becomes 0 since there is no spread in the predictions. To attain an ensemble of forecasts, but keeping the interpolator dropout disabled, we also include an ablation row where we perturb the initial conditions with small random noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon \mathbf{I})$, where we use $\sigma_\epsilon = 0.05$.

Refining the forecasts after cold sampling We find in Table 5 (*No ref.* row), that the addition of line 6 to the cold sampling algorithm (see Alg. 2) can sometimes improve performance. However, this is not consistent across datasets: While for the SST dataset we hardly observe any difference, the scores improve considerably for the spring mesh dataset. A reason could be the relatively long training horizon used for spring mesh (134 for spring mesh versus 16 or 7 for Navier-Stokes or SST). In practice, we recommend practitioners to train DYffusion with the refinement step being disabled in order to accelerate inference time (since the refinement step requires one additional forward pass per output timestep). Then, during evaluation it is encouraged to perform inference with DYffusion with the refinement step being both enabled as well as disabled to analyze whether enabling the refinement step can meaningfully improve the forecasts.

Accelerated Sampling from DYffusion In Fig. 5 we study how sampling from DYffusion can be accelerated in a similar way to how DDIM [60] can accelerate sampling from Gaussian diffusion models. The continuous-time nature of the backbone networks in DYffusion, invites using arbitrary dynamical timesteps as diffusion states at inference time. Thus, to accelerate sampling, we can skip some of the N diffusion steps used for training, $S_{train} = \{i_n\}_{n=0}^{N-1}$, which automatically results in fewer neural network forward passes and thus faster inference. In the simplest case, we can only use the base schedule $S_{base} = \{0, 1, \dots, h-1\}$, where the diffusion states correspond in a one-to-one mapping to the temporal resolution of the dynamical data (see black lines in Fig. 2). In Fig. 5, we start with S_{base} as inference schedule (left-most dots in each subplot) and then incrementally add more diffusion steps from $S_{train} \setminus S_{base}$ to it, until reaching the full training schedule (right-most dots). We find that sampling can be significantly accelerated with marginal drops in CRPS and MSE performance. Note that the dynamical timesteps needed as outputs of DYffusion or for downstream applications pose a lower bound (here, S_{base}) on how much we can accelerate our method, since any such output timestep needs to be included in the sampling schedule or in the set of output timesteps, J (line 6 in Alg. 2).

Choosing the training horizon In any multi-step forecasting model, the training horizon, h, is a key hyperparameter choice. Usually, its choice is constrained by the number of timesteps that fit into GPU memory, and it is expected that larger training horizons will improve performance when evaluated on long (autoregressive) rollouts. However, for continuous-time models including ours, where the number of timesteps needed in GPU memory does not change as a function of h (see Table 7), the choice of h is flexible. In Table 6, we explore using three different training horizons

[4]https://github.com/lucidrains/denoising-diffusion-pytorch/blob/main/denoising_diffusion_pytorch/denoising_diffusion_pytorch.py

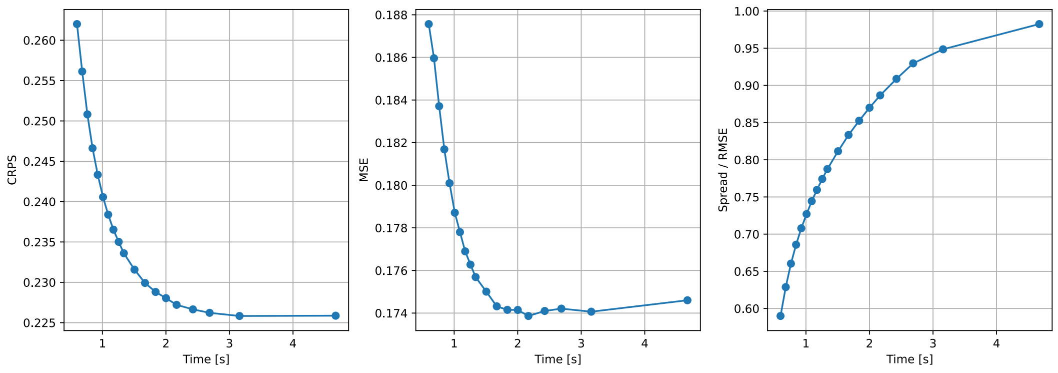


Figure 5: There is a strong trade-off between inference speed (x-axis) and performance (y-axis) as a function of the number of diffusion steps used for inference by DYffusion. Here, we show the SST test scores for one run of DYffusion, which was trained with 35 auxiliary diffusion steps (on top of the 7 given by the data). Each dot from left to right represents performing inference with an increasing number of diffusion steps. Each dot uses the base schedule $S_{base} = \{0, 1, \dots, h-1\}$, where $h = 7$, plus N_{aux} additional diffusion steps drawn from the ones used for training. $N_{aux} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 23, 26, 29, 35)$, and each such additional diffusion step corresponds to an implicit dynamical timestep in $(0, 1)$. Interestingly, almost equivalent (for CRPS) or slightly better (for MSE) scores can be sometimes obtained by using fewer diffusion steps than used for training (right-most dots of each subplot), which immediately benefits inference speed.

Table 6: Navier-Stokes ablation of the training horizon, h. Note that $h = 16$ corresponds to the main results in Table 1, and that $h = 1$ corresponds to a next-step prediction model. All methods are evaluated on the 64-step test trajectories. For example, for $h = 1$ ($h = 16$) the corresponding methods are unrolled autoregressively 64 (4) times.

Method	CRPS	MSE	SSR
Dropout ($h = 1$)	0.132 ± 0.006	0.046 ± 0.006	0.002 ± 0.000
Dropout ($h = 8$)	0.086 ± 0.012	0.026 ± 0.002	0.416 ± 0.293
Dropout ($h = 16$)	0.078 ± 0.001	0.027 ± 0.001	0.715 ± 0.005
Dropout ($h = 32$)	0.078 ± 0.001	0.025 ± 0.001	0.651 ± 0.005
DYffusion ($h = 8$)	0.076 ± 0.002	0.027 ± 0.001	0.701 ± 0.024
DYffusion ($h = 16$)	**0.067** $\pm$ **0.003**	**0.022** $\pm$ **0.002**	**0.877** $\pm$ **0.006**
DYffusion ($h = 32$)	0.075 ± 0.003	0.028 ± 0.001	0.862 ± 0.038

($h \in \{8, 16, 32\}$) for the Navier-Stokes dataset for both the bare-bone time-conditioned Dropout model as well as DYffusion. For DYffusion, this means that we train both an interpolator network as well as a corresponding forecaster net with the same training horizon. Note that $h = 16$ corresponds to the main results. We find that $h = 16$ is a sweet spot for DYffusion. However, any of the used horizons results in better scores than the best baseline (for any baseline training horizon).

B.5.4 Modeling Complexity of DYffusion and baselines

In Table 7 we enumerate the different modeling and compute requirements needed for each of the baselines and our method. Dropout (multi-step) refers to the bare-bone backbone network forecasting all h timesteps $\mathbf{x}_{t+1:t+h}$ in a single forward pass. Dropout (continuous) refers to the bare-bone backbone network forecasting one timestep $\mathbf{x}_{t+k}$ for $k \in \{1, \dots, h\}$ in a single forward pass, conditioned on the time, k. Both methods perform similarly in our exploratory experiments (not shown), and in our experiments we always report the scores of the time-conditioned (i.e continuous) variant. The multi-step approach corresponds to the way the backbone model of a video diffusion model operates, while the continuous variant corresponds to how the forecaster network in DYffusion operates. Video diffusion models have higher modeling complexity because they need to model the full "videos", $\mathbf{x}_{t+1:t+h}$, at each diffusion state (or corrupted versions of it). Especially for long horizons, h, and high-dimensional data with several channels, this complicates the learning task

Table 7: We report the requirements needed to train a method to forecast up to h steps into the future, where c refers to the number of input/output channels (e.g. 1 for SST data), and w to the window size (here, $w = 1$). In the second and third column we report the input and output channel dimensions, respectively, that the (backbone) neural network needs to have (assuming that the window dimension is concatenated to the channel dimension). $|\mathrm{Mem}(\mathbf{x}_t)|$ refers to the number of timesteps that need to be present in (GPU) memory in order to compute the training objective. The last column denotes how many network forward passes are needed to get the forecasts for all h timesteps. Here, N_1, N_2 refer to the number of (sampling) diffusion steps used by DDPM/MCVD and DYffusion, respectively. Usually, $N_1 > N_2 \geq h$, since Gaussian noise diffusion models will require more diffusion steps to attain comparable predictive skill. For Navier-Stokes, $N_2 = h$, for SST $N_2 < 50$ and $N_1 = 1000$ (for MCVD). The factor of 3 for DYffusion is a result of the two extra interpolator network forward passes needed in line 4 of Alg. 2. For large horizons, the model size and memory requirements of multi-step models and conventional diffusion models can be prohibitive. It is clear that (video) diffusion models do not scale well for long horizons.

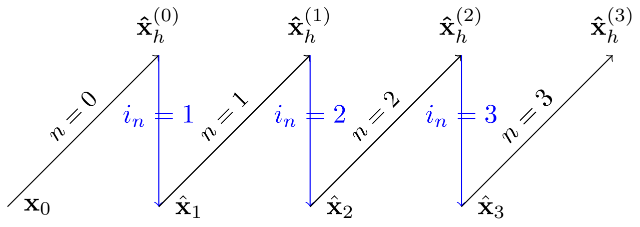
Modeling complexity

Method	c_{in}	c_{out}	$\lvert\mathrm{Mem}(\mathbf{x}_t)\rvert$	#Forward
Dropout (continuous)	$w * c$	c	$w + 1$	h
Dropout (multi-step)	$w * c$	$h * c$	$w + h$	1
DDPM / MCVD	$(h + w) * c$	$h * c$	$w + h$	N_1
DYffusion	$w * c$	c	$w + 1$	$3 * N_2$

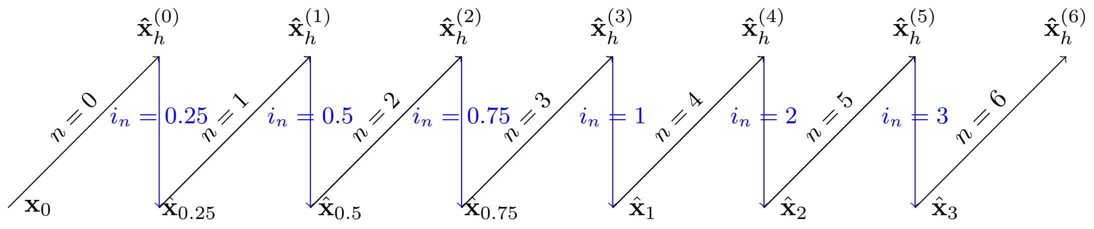
for the neural network. Meanwhile, our method is only slightly impacted by the choice of h (only implicitly through h being some kind of lower bound on the number of diffusion steps in DYffusion).

B.6 Sampling Trajectories

Example sampling trajectories of our approach, as a function of the schedule i_n, are visualized in Fig. 6, where the top one corresponds to the simplest case where we use a one-to-one mapping between diffusion steps, n, and interpolation/dynamical timesteps, i_n. Each of the intermediate $\hat{\mathbf{x}}_i$ can be used as a forecast for timestep i. The forecaster network, F_θ, repeatedly forecasts $\mathbf{x}_h$, but does so with increasing levels of skill (analogously to how conventional diffusion models iteratively denoise/refine the predictions of the "clean" data, $\mathbf{s}^{(0)}$).



(a) Basic schedule, 1-to-1 diffusion step to dynamical timestep mapping



$\hat{\mathbf{x}}_h^{(0)}$
$\hat{\mathbf{x}}_h^{(1)}$
$\hat{\mathbf{x}}_h^{(2)}$
$\hat{\mathbf{x}}_h^{(3)}$
$\hat{\mathbf{x}}_h^{(4)}$
$\hat{\mathbf{x}}_h^{(5)}$
$\hat{\mathbf{x}}_h^{(6)}$
$n = 0$
$i_n = 0.25$
$n = 1$
$i_n = 0.5$
$n = 2$
$i_n = 0.75$
$n = 3$
$i_n = 1$
$n = 4$
$i_n = 2$
$n = 5$
$i_n = 3$
$n = 6$
$\mathbf{x}_0$
$\hat{\mathbf{x}}_{0.25}$
$\hat{\mathbf{x}}_{0.5}$
$\hat{\mathbf{x}}_{0.75}$
$\hat{\mathbf{x}}_1$
$\hat{\mathbf{x}}_2$
$\hat{\mathbf{x}}_3$

(b) Additional diffusion steps (here, 3) map uniformly to $(0, 1)$

Figure 6: Exemplary sampling trajectories of two different schedules for mapping diffusion steps, n, to interpolation (dynamical) timesteps, i_n. The schedules are illustrated using a horizon of $h = 4$. The black lines represent forecasts performed by the forecaster network, F_θ. The first forecast is performed based on the initial conditions, $\mathbf{x}_0$. The blue lines represent the subsequent temporal interpolation performed by the interpolator network, $\mathcal{I}_\phi$.