Predicting North Atlantic Surface Ocean Temperature

Darina Mamyt[1*] and Zihan Yin[1†]

*Corresponding author(s). E-mail(s): dm4037@columbia.edu;
Contributing authors: zy2770@columbia.edu;
†These authors contributed equally to this work.

Abstract

Predicting short term sea surface temperature (SST) is crucial for shipping, ecosystem management and climate research. To explore whether modern machine learning models can learn local ocean dynamics directly from reanalysis data, this project investigates three deep learning approaches—convolutional neural networks (CNNs), long short term memory (LSTM) networks and conditional diffusion models, to forecast the surface temperature of the North Atlantic from the GLORYS12 global ocean reanalysis. Our goal is to compare the capabilities of these models against simple physical baselines and to understand whether including surface horizontal velocity information improves temperature forecasts.

Keywords: Diffusion Model, CNN, LSTM, North Atlantic Ocean, Surface Ocean Termperature

1 Introduction

Predicting short term sea surface temperature (SST) is crucial for shipping, ecosystem management, and climate research [**?**] short term ocean forecast supports upper ocean dynamics, down-scaling of climate modeling, monitoring of marine ecosystems, early detection of marine heatwave conditions, improved initialization for ocean models, coastal ecosystems, and management. Numerical hydrodynamic models provide accurate forecasts but are computationally expensive and require significant infrastructure, limiting their applicability for rapid regional predictions. This makes it attractive to explore data–driven approaches that learn the local SST dynamics directly from existing reanalysis products.

In this project we use the GLORYS12V global ocean reanalysis as a surrogate for the "truth" and ask how well relatively compact deep learning models can forecast the surface temperature of the North Atlantic ocean. We compare three types of models: a convolutional neural network (CNN), a sequence model based on long short–term memory (LSTM), and a conditional diffusion model, against simple physical or statistical baselines. The three prediction tasks are: (i) forecasting temperature from recent temperature only (Temp→Temp), (ii) forecasting horizontal velocity from velocity (Vel→Vel), and (iii) forecasting temperature from both temperature and velocity (Temp+Vel→Temp). Throughout, we focus on a 7–day forecast horizon, which is long enough for model differences to appear, but short enough that the dynamics are still dominated by mesoscale variability rather than long–term climate trends.

2 DATASET DESCRIPTION

2.1 Data source and domain

All data come from the GLORYS12V global ocean reanalysis produced by the Copernicus Marine Environment Monitoring Service (CMEMS). GLORYS12V is based on a global ocean general circulation model constrained by a wide range of satellite and in–situ observations and provides a dynamically consistent estimate of the ocean state. We extract a fixed two–dimensional patch in the North Atlantic, located off the east coast of North America, and treat each daily analysis as a gridded snapshot on this regional domain. The time period considered in this study spans ten years (2014–2024), yielding 3654 daily fields.

For each day and each grid point within the patch we use the following variables:

- Sea–surface temperature T (GLORYS variable `thetao`), used as both input and prediction target.
- Surface horizontal velocity components u and v (GLORYS variables `uo` and `vo`), used as inputs and prediction targets for the velocity tasks, and as additional predictors for the Temp+Vel→Temp task.
- Time of year, encoded via the day–of–year index and transformed into "seasonality" features for the diffusion model.

Land points within the regional domain are masked out using the GLORYS land/sea mask; only ocean grid cells are considered during training and evaluation.

2.2 Preprocessing

We apply a series of preprocessing steps before constructing the supervised learning datasets. First, we crop the chosen North Atlantic patch from the full GLORYS grid and, for computational reasons, downsample the fields from 256×256 to 128×128 using spatial averaging. This resolution is a compromise between capturing mesoscale structures and fitting batches of data into GPU memory.

Second, we normalise each physical variable using standard z–score normalisation. For temperature and each velocity component we compute the mean μ and standard

deviation σ over the training portion of the dataset and those estimated mean and variance applied to the validation and test data to avoid techncial data leakage.

Third, we construct a static land mask. Although the GLORYS NaN mask can differ slightly between variables and time steps, we find that the differences over our regional patch amount to only 6 pixels. We therefore take the union of all NaN locations across time and variables and define a single binary mask that identifies ocean grid points. During training and evaluation, losses are computed only over ocean points, so that land pixels do not influence either optimization or reported metrics.

Finally, we build explicit time–of–year encodings for the diffusion model. For each day index h with day–of–year $d(h) \in \{1, \dots, 366\}$ we define

$$\alpha(h) = 2\pi \frac{d(h) - 1}{365}, \qquad \sin_{\mathrm{DOY}}(h) = \sin\big(\alpha(h)\big), \qquad \cos_{\mathrm{DOY}}(h) = \cos\big(\alpha(h)\big),$$

and broadcast these scalar values to full $H \times W$ maps. These two channels are concatenated with the physical fields and allow the diffusion model to distinguish different phases of the seasonal cycle even though the input and output temperatures have been normalized. The full time series is then partitioned into contiguous training, validation, and test segments in a 70/15/15 split, respecting temporal order to avoid information leakage from future to past.

3 Deterministic CNN Model

3.1 Architecture

We use a lightweight fully-convolutional network (no pooling and no fully-connected layers), so the spatial resolution is preserved throughout the model. Different forecasting tasks instantiate the same backbone with different input/output channel counts $(C_{\mathrm{in}}, C_{\mathrm{out}})$ and hidden width (H_d).

Given an input tensor $x \in \mathbb{R}^{B \times C_{\mathrm{in}} \times H \times W}$, the CNN predicts

$$\hat{y} = f_{\mathrm{CNN}}(x) \in \mathbb{R}^{B \times C_{\mathrm{out}} \times H \times W}.$$

The backbone consists of four 3×3 convolutional layers with padding 1 (to keep $H\times W$ unchanged) and ReLU activations after the first three convolutions:

$$f_{\text{CNN}} = \begin{cases} \text{Conv}_{3\times3}(C_{\text{in}} \to H_d) \\ \downarrow \\ \text{ReLU} \\ \downarrow \\ \text{Conv}_{3\times3}(H_d \to H_d) \\ \downarrow \\ \text{ReLU} \\ \downarrow \\ \text{Conv}_{3\times3}(H_d \to H_d) \\ \downarrow \\ \text{ReLU} \\ \downarrow \\ \text{Conv}_{3\times3}(H_d \to C_{\text{out}}) \end{cases}$$

Here H_d controls the model capacity (e.g., we evaluate multiple widths such as 16/32/64). Because all layers are local convolutions, the network learns spatially coherent corrections while remaining parameter-efficient and easy to train. Also layer 3 is optional, based on the performance of different layer number we will decide whether to drop it or not in the hyperparameter search part later. Also, we put an early stopper in the architecture, if there is more than 20 epochs while loss not decreases for at least 0.001, the training will stop.

4 LSTM Model

4.1 Architecture

We use a pixel-wise tokenized LSTM forecaster. Given an input tensor $X \in \mathbb{R}^{B\times C_{\text{in}}\times H\times W}$, we first flatten the spatial grid into a sequence of length $L = H \cdot W$ and treat the channel vector at each grid cell as a token:

$$X \;\to\; \tilde{X} \in \mathbb{R}^{B\times L\times C_{\text{in}}}, \quad \tilde{X}_{b,\ell,:} = X_{b,:,i,j}.$$

The resulting sequence is fed into a multi-layer LSTM (hidden size H_d, number of layers N_L):

$$H = \text{LSTM}(\tilde{X}) \in \mathbb{R}^{B\times L\times H_d}.$$

A linear projection is then applied to each token state to produce the output channels, and the sequence is folded back to the original spatial layout:

$$Y = \text{reshape}(\text{Linear}(H)) \in \mathbb{R}^{B\times C_{\text{out}}\times H\times W}.$$

Here, C_{in} and C_{out} are configured per task (e.g., temperature or velocity prediction). Also, we put an early stopper in the architecture, if there is more than 20 epochs while loss not decreases for at least 0.001, the training will stop.

Loss Function (Deterministic CNN/LSTM)

We use a masked mean squared error (MSE) over valid ocean points. Let the prediction be $\hat{Y} \in \mathbb{R}^{B\times C_{\text{out}}\times H\times W}$, the target be $Y \in \mathbb{R}^{B\times C_{\text{out}}\times H\times W}$, and the binary mask be $M \in \{0,1\}^{1\times 1\times H\times W}$. The temperature loss is:

$$\mathcal{L}_{\text{temp}} = \frac{\sum_{b,c,i,j} M_{i,j}\left(\hat{Y}_{b,c,i,j} - Y_{b,c,i,j}\right)^2}{\sum_{i,j} M_{i,j}}.$$

For velocity prediction ($C_{\text{out}} = 2$), we add a divergence regularization term to encourage physically consistent flow fields. With $\hat{\mathbf{u}} = (\hat{u}, \hat{v})$, the loss is:

$$\mathcal{L}_{\text{vel}} = \mathcal{L}_{\text{mse}} + \lambda_{\text{div}} \frac{\sum_{b,i,j} M_{i,j}\left(\nabla\cdot\hat{\mathbf{u}}_{b,i,j}\right)^2}{\sum_{i,j} M_{i,j}},$$

where λ_{div} is the divergence weight and $\nabla\cdot\hat{\mathbf{u}}$ is computed via finite differences. The three models optimize the same objective—minimizing a masked mean-squared error over valid ocean pixels (with an additional divergence regularization term for the velocity task). However, their loss *forms* differ because the output parameterizations differ: CNN/LSTM directly regress the target field, whereas the diffusion model predicts noise (and reconstructs the field), leading to different mathematical expressions of the loss. The loss function for diffusion model will be discussed later.

5 DIFFUSION MODEL

5.1 Architecture

To model the 7-day forecast of surface temperature and velocity fields, we use a **conditional 2D denoising diffusion model**. The goal is to learn the conditional distribution

$$p\big(y_{t+7} \mid x_t\big),$$

where y_{t+7} is the target field (temperature or velocity) at $t+7$ days, and x_t collects the fields available at time t (and, for temperature, also $t-1$) together with simple time encodings.

For the temperature task, the conditioning tensor has

$$C_{\text{cond}} = 4$$

channels: temperature at $t-1$ and t, and two seasonal encodings (sine and cosine of day-of-year). The target tensor is the temperature at $t+7$ with

$$C_{\text{tar}} = 1$$

channel. In a mini-batch we thus work with

$$x_{\text{cond}} \in \mathbb{R}^{B\times C_{\text{cond}}\times H\times W}, \qquad y \in \mathbb{R}^{B\times C_{\text{tar}}\times H\times W}.$$

During training we pad all tensors so that the spatial sizes $(H_{\text{pad}}, W_{\text{pad}})$ are divisible by the downsampling factor of the UNet and crop back to (H, W) for evaluation.

5.2 UNet2D architecture

The noise-prediction network is a 2D UNet from the `diffusers` library:

- Input channels:
$$C_{\text{in}} = C_{\text{cond}} + C_{\text{tar}} = 5,$$
given by concatenating x_{cond} and the noised target x_t along the channel dimension.
- Output channels:
$$C_{\text{out}} = C_{\text{tar}} = 1,$$
corresponding to an estimate $\hat{\varepsilon}_\theta$ of the Gaussian noise added to the target.
- Depth and width: a three-level encoder–decoder with
$$\texttt{block\_out\_channels} = (64, 128, 128),$$
and two convolutional layers per level. This provides a multi-scale representation with increasing receptive field.
- No attention is used in this configuration (pure convolutional UNet).

$$\text{Vel} \to \text{Vel} \quad \text{as} \quad \big(u(h), v(h)\big) \;\mapsto\; \big(u(h+7), v(h+7)\big).$$

The UNet maps

$$x_{\text{in}} \in \mathbb{R}^{B\times 5\times H_{\text{pad}}\times W_{\text{pad}}} \quad\longrightarrow\quad \hat{\varepsilon}_\theta \in \mathbb{R}^{B\times 1\times H_{\text{pad}}\times W_{\text{pad}}}.$$

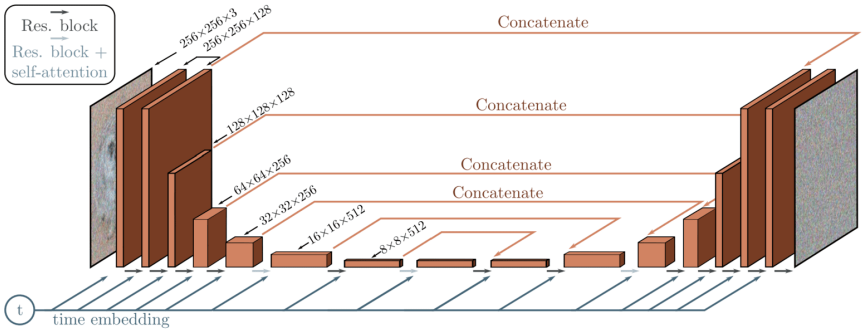


Fig. 1 UNet2D architecture.

5.3 Diffusion process and losses

We use a standard DDPM forward process.

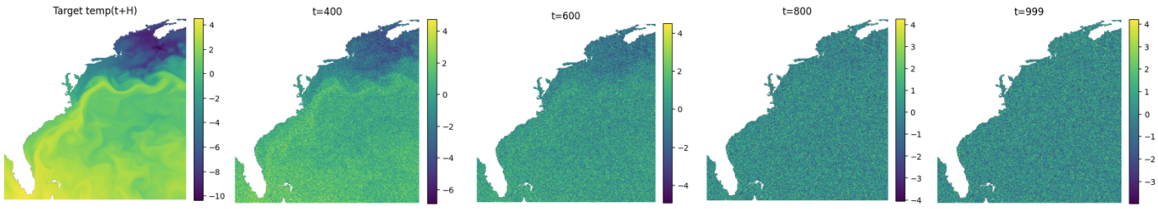


Fig. 2 Forward diffusion process for a fixed target temperature field: as t increases, more Gaussian noise is added until the field approaches pure noise.

For a random time step t and Gaussian noise $\varepsilon \sim \mathcal{N}(0, I)$, the noised target is

$$x_t = \sqrt{\bar{\alpha}_t}\, y + \sqrt{1 - \bar{\alpha}_t}\, \varepsilon,$$

where $\bar{\alpha}_t = \prod_{s=1}^{t}(1 - \beta_s)$ is computed from the scheduler's β_t schedule.

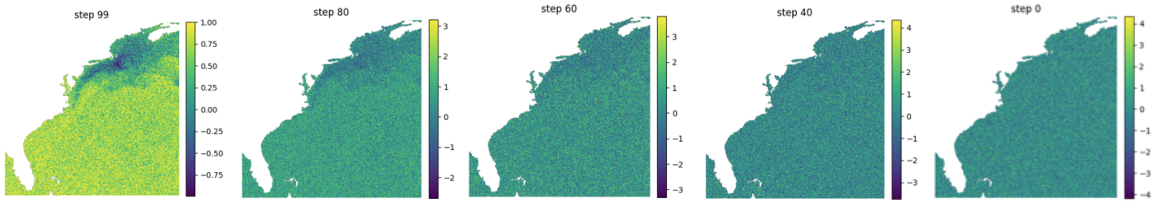


Fig. 3 Backward diffusion process (reverse denoising): starting from pure noise at step 0, the model gradually reconstructs a realistic temperature field as the reverse diffusion steps progress toward step 99.

Temperature loss.

For temperature, the training objective is a masked mean squared error between predicted and true noise:

$$\mathcal{L}_{\text{temp}}(\theta) = \mathbb{E}_{t,\varepsilon,x_{\text{cond}},y}\Big[\text{MSE}_{\text{ocean}}\big(\hat{\varepsilon}_\theta(x_t, t, x_{\text{cond}}), \varepsilon\big)\Big],$$

where the MSE is computed only over ocean pixels using the GLORYS land mask.

Velocity loss with divergence penalty.

For velocity experiments, we additionally encourage incompressibility. After predicting $\hat{\varepsilon}_\theta$ we reconstruct a denoised estimate $\hat{y}_0$ via

$$\hat{y}_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\,\hat{\varepsilon}_\theta}{\sqrt{\bar{\alpha}_t}}.$$

Writing $\hat{\mathbf{u}}_0 = (u, v)$ for the horizontal velocity components, we approximate the divergence by finite differences and define

$$\nabla \cdot \hat{\mathbf{u}}_0 \approx \frac{\Delta u}{\Delta x} + \frac{\Delta v}{\Delta y},$$

$$\mathcal{L}_{\text{div}} = \left\| \nabla \cdot \hat{\mathbf{u}}_0 \right\|^2_{\text{ocean}}.$$

The total loss for velocity becomes

$$\mathcal{L}_{\text{vel}}(T) = \mathcal{L}_\varepsilon(T) + \lambda_{\text{div}}\,\mathcal{L}_{\text{div}}(T),$$

with λ_{div} a small divergence penalty weight. For the temperature task we set $\lambda_{\text{div}} = 0$.

5.4 velocity-to-velocity

For the **velocity-to-velocity** task we predict the 7-day-ahead horizontal velocity field from the present velocity only. Let $u(h)$ and $v(h)$ denote the zonal and meridional surface velocity components at day h. The input and target for this task are

$$x_h^{(\text{vel})} = \big[\, u(h),\ v(h) \big], \qquad y_h^{(\text{vel})} = \big[\, u(h+7),\ v(h+7) \big],$$

so that the model learns to propagate the two-component horizontal velocity field forward in time without explicit temperature information.

5.5 temperature-and-velocity-to-temperature

For the **temperature-and-velocity-to-temperature** task we combine both sources of information: recent temperature, current velocity and seasonality, and again predict the temperature 7 days ahead. Using the same notation as above, we define the input as

$$x_h^{(\text{temp+vel})} = \big[\, \sin_{\text{DOY}}(h),\ \cos_{\text{DOY}}(h),\ T(h-1),\ T(h),\ u(h),\ v(h) \big],$$

and the target as

$$y_h^{(\text{temp+vel})} = T(h+7).$$

This third dataset allows us to assess how much additional predictive skill is gained by providing the model with the surface flow field in addition to recent temperature and seasonal phase.

6 HYPERPARAMETERS SEARCH

6.1 Hyperparameters for CNN: Grid search

For each hyperparameter configuration, we perform a short training run for model selection. Specifically, we set the search training length to `SEARCH_EPOCHS`=30 and use a reduced *search dataset* consisting of a 20×20 spatial subregion and a 3-year time window (1095 daily samples). For each configuration, we compute the validation RMSE and select the best-performing setting. The same search protocol is applied to Tasks 2 and 3 for both CNN and LSTM models to ensure a consistent comparison.

Task: Temp→Temp.

For the CNN temperature forecasting task, we use grid search over the following dimensions:

- **Network depth:** 3-layer vs. 4-layer CNN. We evaluate both depths independently and then compare their best results to determine the preferred architecture.
- **Hidden width (H_d):** {16, 32, 64}.
- **Learning rate (η):** $\{10^{-4}, 3 \times 10^{-4}, 10^{-3}\}$.

Here are the RMSE tables for hyperparameter search, the first one is for 3 layers, the second one is for 4 layers.

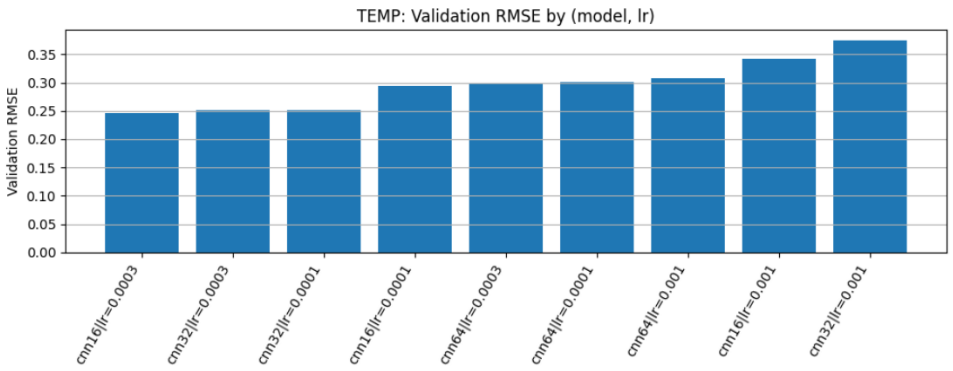


Fig. 4 Validation search RMSE For CNN 3L model in task t to t.

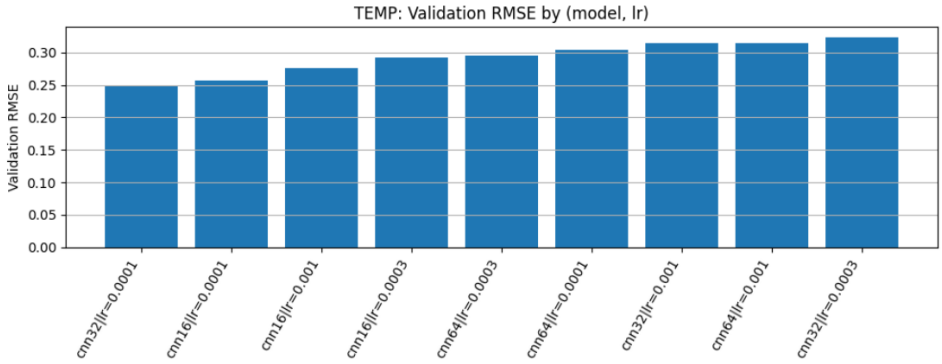


Fig. 5 Validation search RMSE For CNN 4L model in task t to t.

Best RMSE : 0.2475.
Here is the parameter search table: The best parameter combination is a **4-layer** CNN with hidden size 32 and learning rate 10^{-4}, achieving a validation RMSE of 0.2475.

Number of Layers	Learning Rate	Hidden Size
3	3×10^{-4}	16
4	10^{-4}	32
	10^{-3}	64

Table 1 Grid search settings for CNN hyperparameters. The highlighted row indicates the best configuration.

Task: Vel→Vel.

For the CNN velocity forecasting task, we perform a grid search over the following dimensions:

- **Network depth:** 3-layer vs. 4-layer CNN.
- **Hidden width (H_d):** {16, 32, 64}.
- **Learning rate (η):** $\{10^{-4}, 3 \times 10^{-4}, 10^{-3}\}$.
- **Divergence penalty (λ_{div}):** $\{0, 10^{-3}, 10^{-2}\}$.

Compared with Task 1, we additionally include a divergence penalty term to discourage the model from producing noisy, locally oscillatory velocity fields and to improve generalization beyond the training set. The validation RMSE values for all combinations are summarized in two tables: the first corresponds to the 3-layer CNN and the second corresponds to the 4-layer CNN.

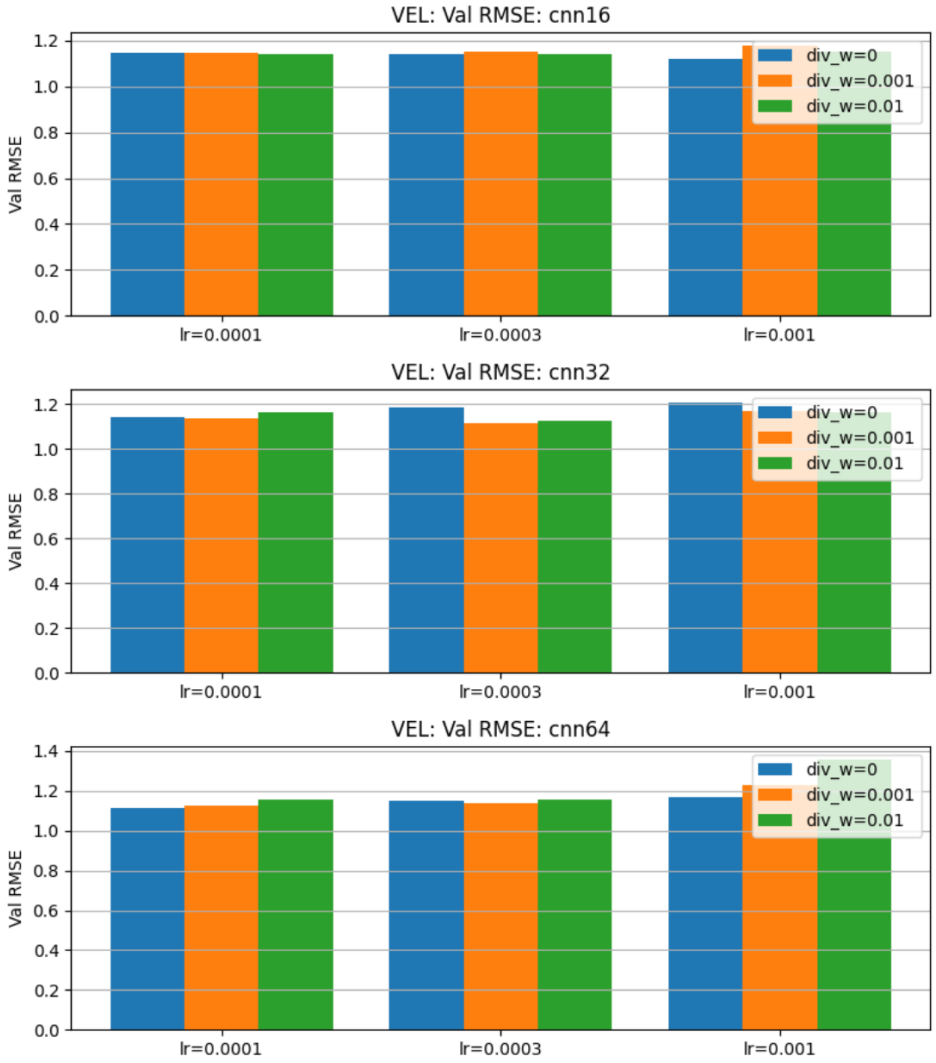


Fig. 6 Validation search RMSE For CNN 3L model in task vel to vel.

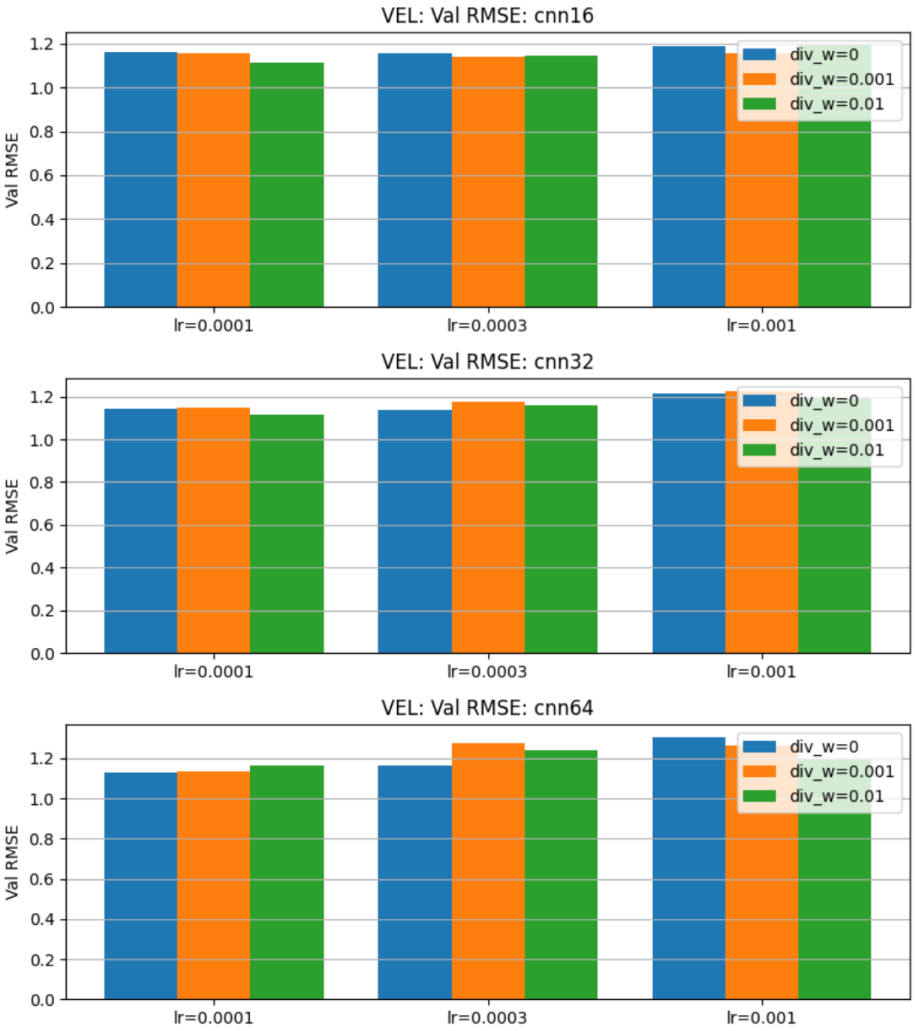


Fig. 7 Validation search RMSE For CNN 4L model in task vel to vel.

Best RMSE : 0.2475.

Here is the parameter search table:

The best parameter combination is 4 convolution layers with 16 hidden size, 1e-4 learning rate, and divergence penalty equal to 0.01. The search RMSE is 1.1127.

Number of Layers	Learning Rate	Hidden Size	Divergence Penalty
3	3×10^{-4}	32	0
4	10^{-4}	16	10^{-2}
	10^{-3}	64	10^{-3}

Table 2 Grid search settings for CNN hyperparameters. The highlighted row indicates the best configuration.

Task: Temp+Vel→Temp.

For the CNN temperature forecasting task with velocity conditioning, we conduct a grid search over the following dimensions:

- **Network depth:** 3-layer vs. 4-layer CNN.
- **Hidden width (H_d):** {24, 48, 96}.
- **Learning rate (η):** $\{10^{-4}, 3 \times 10^{-4}, 10^{-3}\}$.

Here are the RMSE tables for hyperparameter search, the first one is for 3 layers, the second one is for 4 layers.

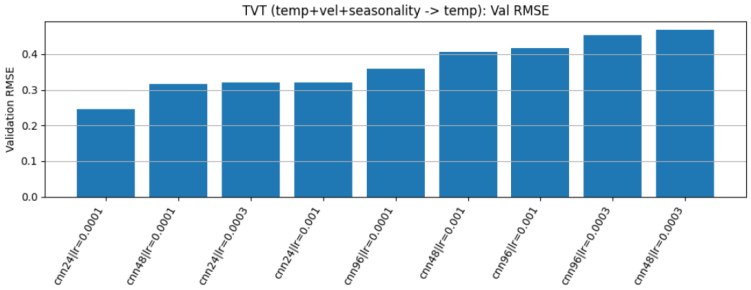


Fig. 8 Validation search RMSE For CNN 3L model in task t + vel to vel.

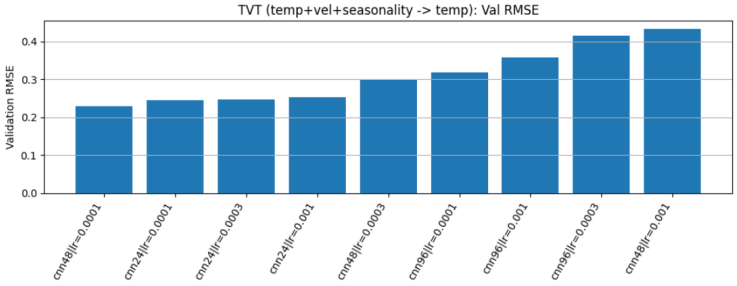


Fig. 9 Validation search RMSE For CNN 4L model in task t + vel to vel.

Here is the parameter search table:
The best parameter combination is 4 convolution layers with 48 hidden size, 1e-4 learning rate. The search RMSE is 0.2295.

Number of Layers	Learning Rate	Hidden Size
3	3×10^{-4}	24
4	10^{-4}	48
	10^{-3}	96

Table 3 Grid search settings for CNN hyperparameters (Temp+Vel→Temp). The highlighted row indicates the best configuration.

6.2 Hyperparameters for LSTM: Grid Search

Task: Temp→Temp.

The search dimensions include the following:

- **Number of layers:** 1 / 2.
- **Hidden size:** 32 / 64 / 128.
- **Learning rate (η):** $\{10^{-3}, 3 \times 10^{-3}, 10^{-4}\}$.

Here is the RMSE table for hyperparameter search.

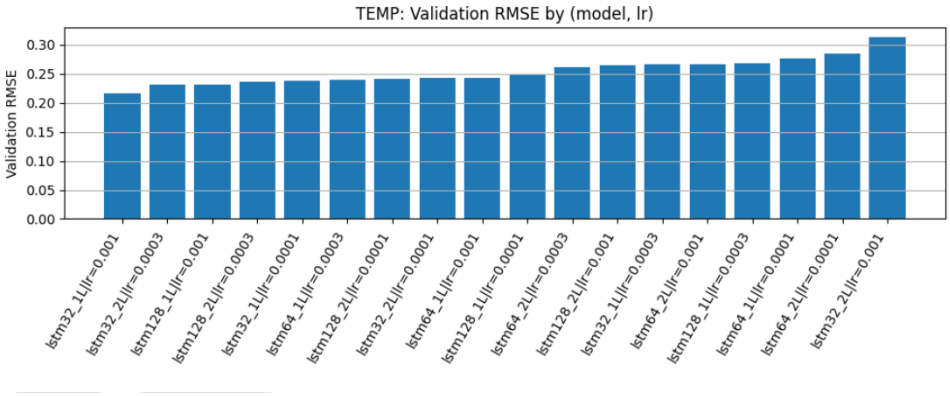


Fig. 10 Validation search RMSE For LSTM model in task t to t.

The best parameter combination is 2 stack layers with 32 hidden size, 1e-3 learning rate. The best search RMSE is 0.2168.

Number of Layers	Learning Rate	Hidden Size
2	10^{-3}	32
1	3×10^{-4}	64
	10^{-4}	128

Table 4 Grid search settings for LSTM hyperparameters. The highlighted row indicates the best configuration.

Task: Vel→ Vel.

For the LSTM velocity forecasting task, we perform a grid search over the following dimensions:

- **Number of layers:** {1, 2}.
- **Hidden size:** {32, 64, 128}.
- **Learning rate (η):** $\{10^{-3}, 3 \times 10^{-3}, 10^{-4}\}$.
- **Divergence penalty (λ_{div}):** $\{0, 10^{-3}, 10^{-2}\}$.

Here is the RMSE table:

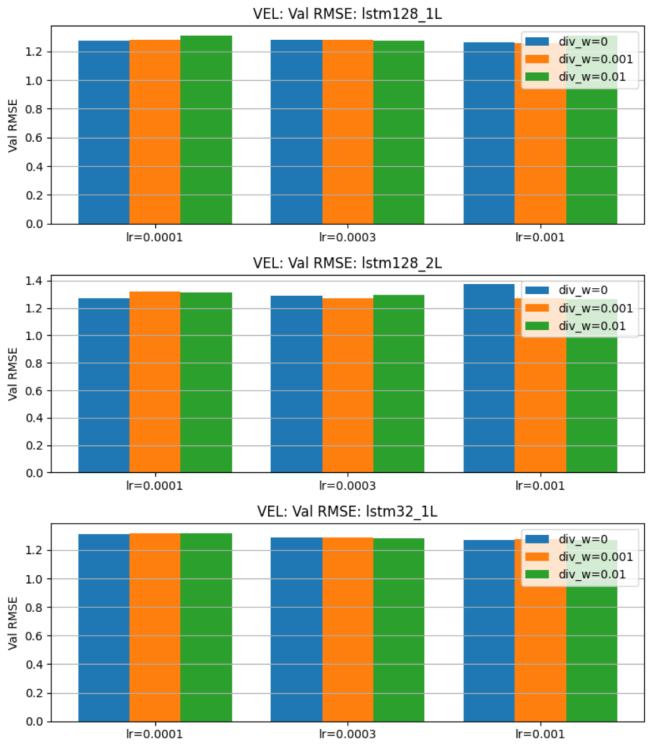


Fig. 11 Validation search RMSE For LSTM model in task vel to vel.

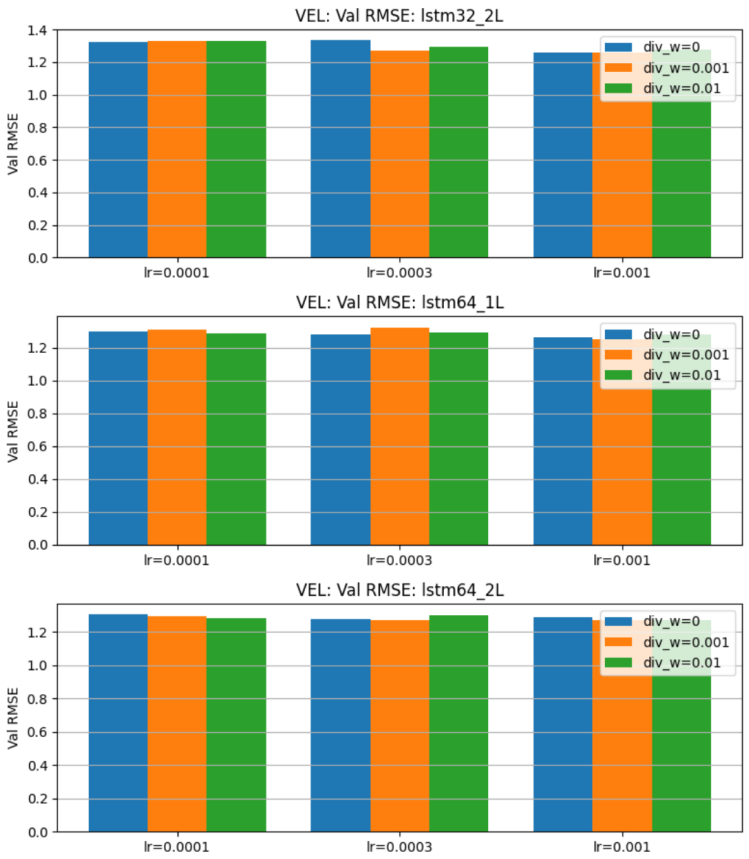


Fig. 12 Validation search RMSE For LSTM model in task vel to vel.

Number of Layers	Learning Rate	Hidden Size	Divergence Penalty
1	10^{-4}	16	0
2	3×10^{-4}	32	10^{-2}
	10^{-3}	64	10^{-3}

Table 5 Grid search settings for LSTM hyperparameters (Vel→Vel). Highlighted cells indicate the selected configuration.

The best parameter combination is 1 layer with 64 hidden size, 1e-3 learning rate and 1e-3 divergence penalty. The best search RMSE is 1.2524.

Task: Temp + Vel → Vel.

For the Temp + Vel → Vel task, we perform a grid search over the following dimensions:

- **Stack layers:** 1-layer vs. 2-layer.
- **Hidden size (H_d):** $\{48,\ 96\}$.
- **Learning rate (η):** $\{10^{-4},\ 3\times10^{-4},\ 10^{-3}\}$.

Here is the RMSE table:

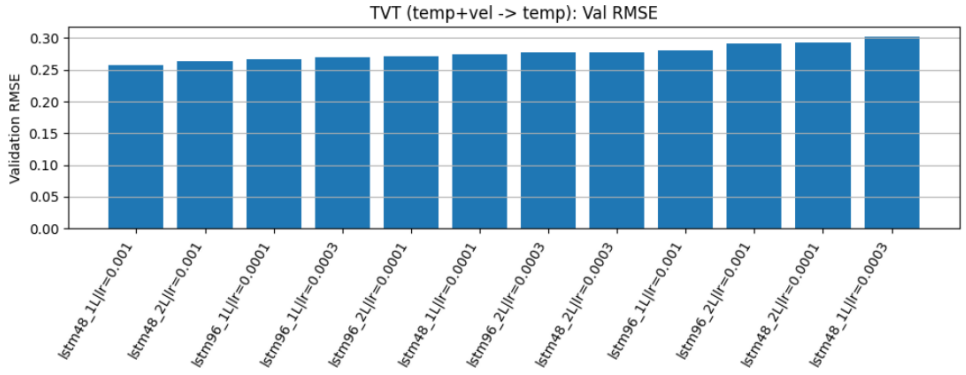


Fig. 13 Validation search RMSE For LSTM model in task vel + t to t.

Number of Layers	Learning Rate	Hidden Size
1	1e-4	48
2	3e-4	96
	1e-3	

Table 6 Grid search settings for LSTM hyperparameters (Temp+Vel → Vel). The highlighted cells indicate the selected configuration.

The best parameter combination is 1 layer with 48 hidden size, 1e-3 learning rate. The best search RMSE is 0.2581.

6.3 Hyperparameters for the DIFFUSION MODEL: Manual search

Because each diffusion model is relatively expensive to train, we did not perform a full grid search. Instead, we carried out a manual hyperparameter search, using short runs (about 10–15 epochs) to compare validation RMSE and then training only the best candidate for a longer period (50-400 epoch with early stopping). The main parameters varied were: UNet depth and width (`block_out_channels`), learning rate, dropout, the number of inference steps, the choice of diffusion scheduler, and the β_t schedule.

This configuration defines a medium-capacity conditional UNet diffusion model conditioned on two days of temperature and seasonal time encodings, trained with a

Table 7 Summary of manual hyperparameter search for the temperature diffusion model.

Learning rate	**UNet block channels**	**β schedule**	**Scheduler**	**Dropout**
1×10^{-4}	$(32, 32)$	scaled linear	DDIMScheduler	0.20
3×10^{-4}	$(64, 128)$	sigmoid	DPMSolverMultistepScheduler	0.10
5×10^{-4}	$(64, 128, 128)$	squaredcos_cap_v2	DDPMScheduler	0.00
	$(64, 128, 256)$	linear		0.05

Table 8 The best hyperparameters for the diffusion model.

Hyperparameter	**Value**	**Description**
Conditioning channels	$C_{\text{cond}} = 4(temp), 2(vel), 6(temp + vel)$	$T_{t-1}, T_t, \sin(\text{DOY}), \cos(\text{DOY})$
Target channels	$C_{\text{tar}} = 1$	Surface temperature at $t + 7$
UNet block widths	(64, 128, 128)	`block_out_channels`
Attention	`False`	No self-attention blocks
Training diffusion steps	1000	`num_train_timesteps`
Inference diffusion steps	100	`num_inference_steps`
Noise schedule type	`linear`	`beta_schedule`
Noise schedule range	$10^{-4} \rightarrow 2 \times 10^{-2}$	$\beta_{\text{start}}, \beta_{\text{end}}$
Scheduler	`DDPMScheduler`	DDPM ancestral sampler
Learning rate	3×10^{-4}	Adam optimizer
Dropout	0.05	UNet dropout probability
Divergence weight	0	λ_{div} (temp only)

linear DDPM schedule and evaluated with 100 sampling steps. It serves as our primary diffusion baseline for the 7-day surface temperature forecast.

7 RESULTS

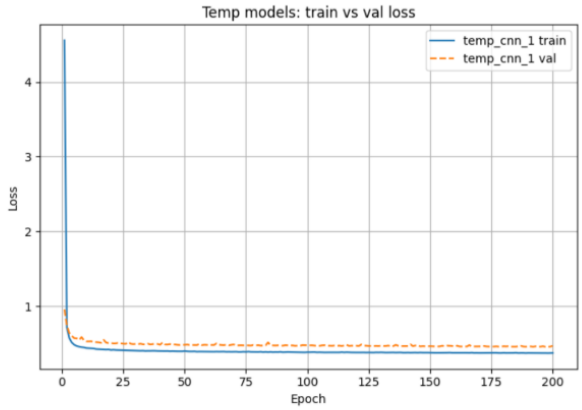
7.1 Quantitative performance

7.1.1 CNN Result on Test Set

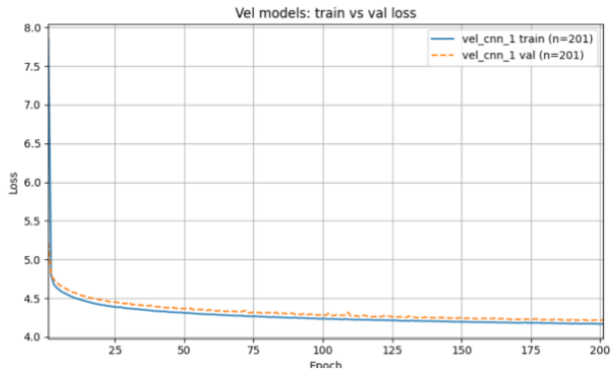
	Task 1	**Task 2**	**Task 3**
RMSE (normalized)	0.6560	2.0288	0.6296
Denormalized result	0.9602 °C	0.2502 m/s	0.9215 °C

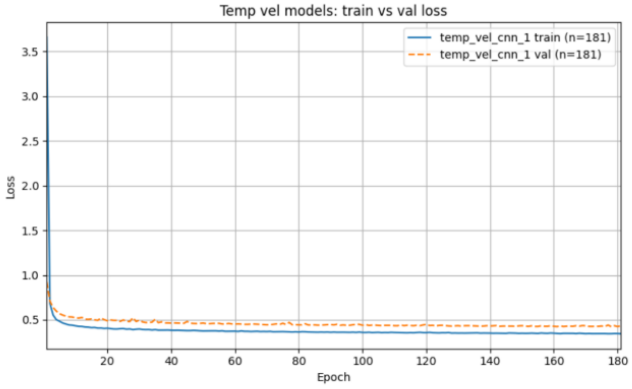
Table 9 Test-set performance summary for the three tasks.

The results show that the CNN model performs well in temperature prediction, and its performance improves after incorporating velocity field information, indicating that the velocity field provides additional useful information. However, using CNN to directly predict the velocity field performs poorly, perhaps because the velocity field itself is too complex, with too many influencing factors, and using velocity alone to predict velocity does not yield good results. Here is the Loss vs. epoch chart for final training.



(NOTE:Early stop at epoch 160)





(Note: Early stop at epoch 181)

Fig. 14 CNN Loss vs. epoch.

7.1.2 LSTM Result on Test Set

	Task 1	**Task 2**	**Task 3**
RMSE (normalized)	0.6685	2.1936	0.6700
Denormalized result	0.9784 °C	0.2705 m/s	0.9807 °C

Table 10 Final test-set RMSE and denormalized errors for the three tasks.

The results show that the LSTM model performed reasonably well in tasks 1 and 3, but its performance in predicting the velocity field was poor. Furthermore, the LSTM model did not show any performance improvement when provided with the velocity field as additional information. Here is the Loss vs. epoch chart for final training.

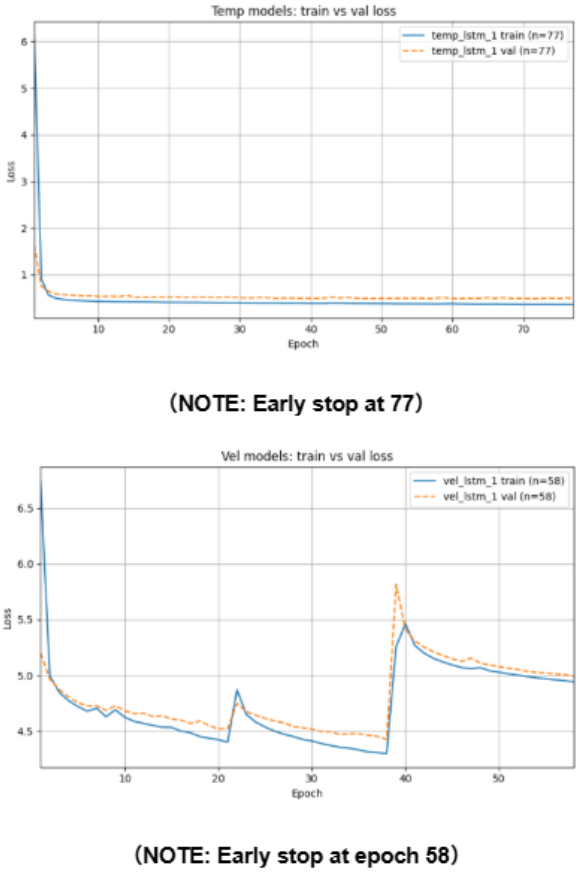


Fig. 15 LSTM Loss vs. epoch.

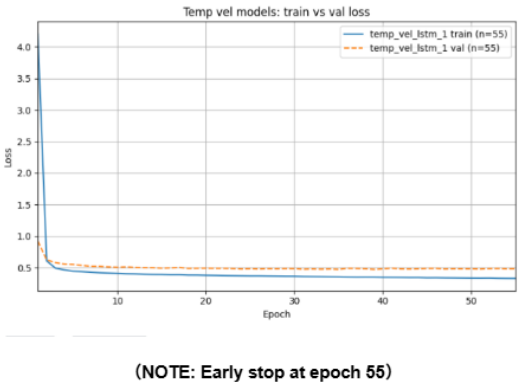


Fig. 16 LSTM Loss vs. epoch.

Figure 15 shows an overall decreasing trend in both training and validation losses, but with two noticeable spikes (around epoch ∼22 and ∼39). Such behavior typically indicates *optimization instability* rather than a simple overfitting pattern. A few likely causes are: (i) **learning rate too large** for the LSTM optimizer dynamics, which can lead to occasional overshooting and sudden loss jumps (especially with Adam when the adaptive moments change quickly); (ii) **gradient explosion in recurrent networks**, where the hidden-state recurrence amplifies gradients and produces abrupt updates unless gradient clipping is used; (iii) **small batch size / noisy gradients** (e.g., $B = 4$), which increases stochasticity and can trigger intermittent large parameter updates; (iv) **non-stationary target scale across time** (even after normalization), where certain time windows contain more energetic flows, making some mini-batches substantially harder and causing temporary spikes. After the large jump near epoch ∼39, both curves settle into a higher-loss regime, suggesting the optimizer may have moved to a worse basin; early stopping at epoch 58 prevents further degradation. In practice, this can often be mitigated by reducing the learning rate, enabling gradient clipping (e.g., $\|\nabla\|_2 \leq 1$), and/or using a learning-rate scheduler.

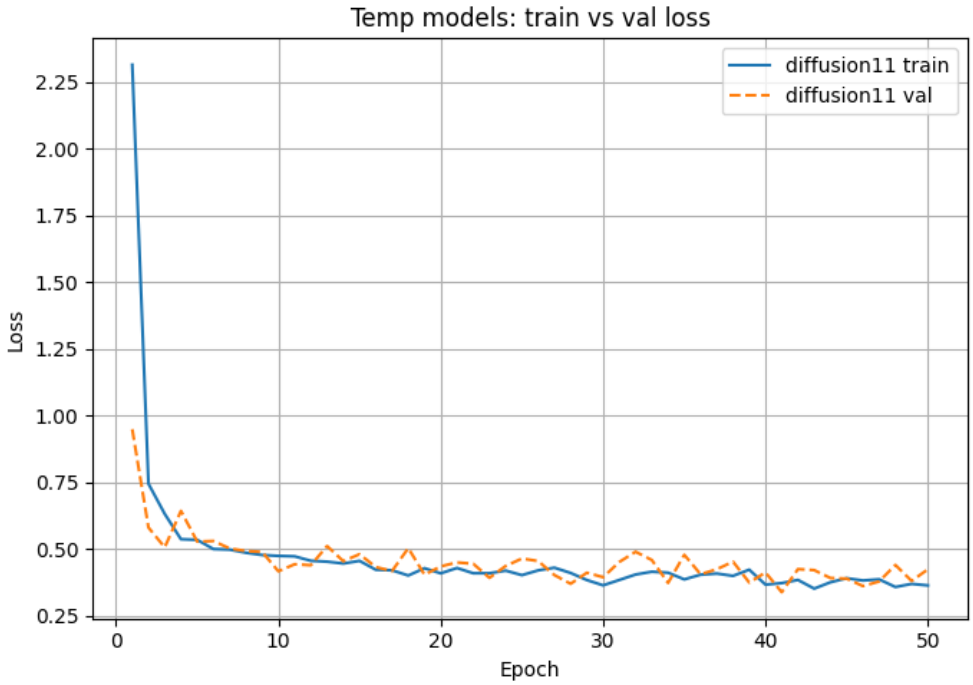


Fig. 17 Diffusion (temperature) loss vs. epoch.

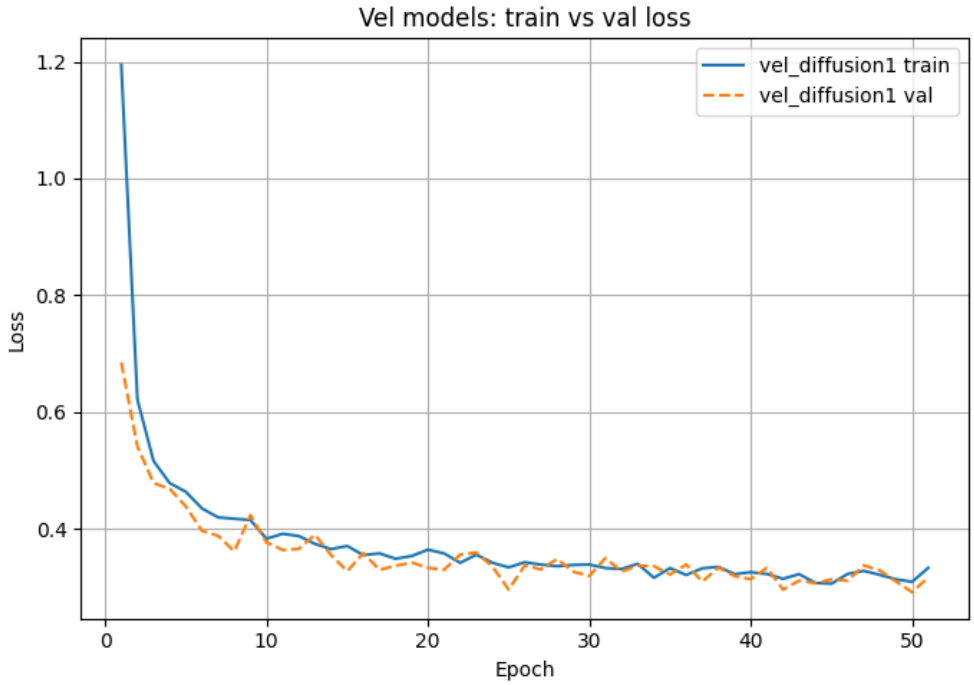


Fig. 18 Diffusion (velocity) loss vs. epoch.

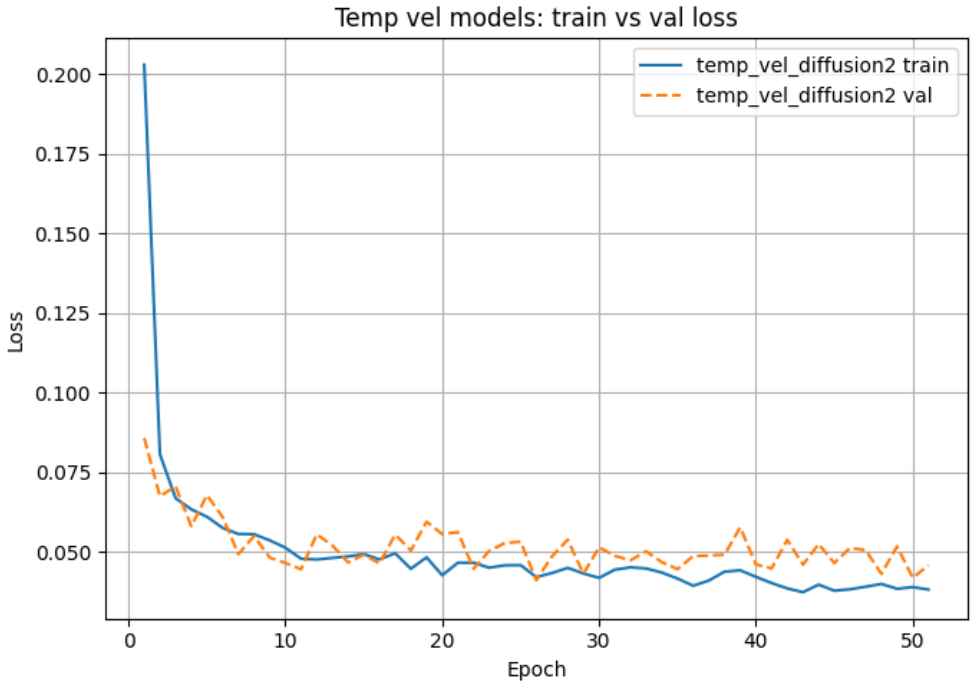


Fig. 19 Diffusion (temperature via velocity) loss vs. epoch.

7.1.3 Diffusion model Result on Test Set

For all three prediction tasks we trained conditional diffusion models based on the UNet2D architecture with the manually tuned hyperparameters summarized in Table 8. In each case the training loss is the masked MSE between the predicted noise and the true Gaussian noise, averaged over ocean points only. The loss curves in Figs.17,18,19, show a rapid decrease during the first few epochs followed by a slow plateau, with training and validation losses tracking each other closely. This indicates that, at the chosen model size, the diffusion models do not strongly overfit the GLORYS patch but instead saturate at a level where further improvement would likely require either more capacity, longer training, or a different loss formulation.

When we convert the final denoised samples back to physical temperature and velocity fields and compute RMSE on the held-out test set, the diffusion models consistently underperform the deterministic CNN baselines. For the Temp→Temp task, the best configuration (`diffusion11`) achieves a test RMSE that is larger than that of the best CNN. The Temp+Vel→Temp diffusion model shows a similar pattern: it converges stably in terms of noise-prediction loss, but its forecast RMSE remains higher than the corresponding CNN trained on the same inputs. For the Vel→Vel task, the diffusion model again fails to match the CNN in RMSE, although the gap is somewhat smaller than in the temperature case.

7.2 Qualitative evaluation

Beyond quatitative metrics, we also visualize predicted temperature maps for several test dates and compare them to the GLORYS ground truth. Even though the final RMSE is worse than that of the CNN, the generated temperature samples are visually realistic and respect the overall geometry of the North Atlantic circulation, albeit with noticeable noise. For velocity prediction, the generated images roughly follow the main flow patterns, but they exhibit much higher contrast than the reference fields.

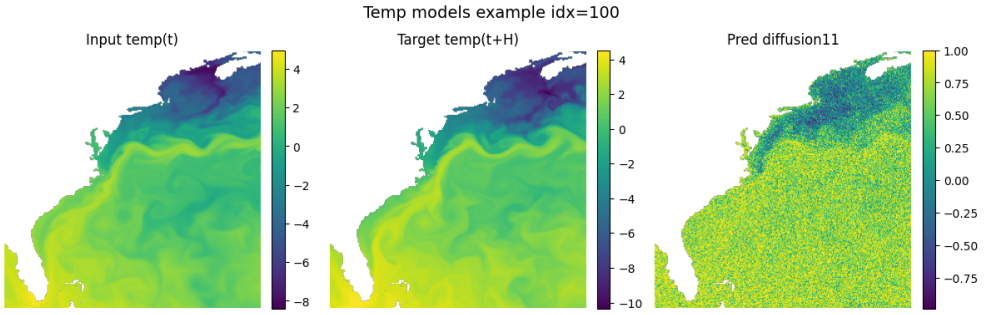


Fig. 20 Diffusion temperature prediction.

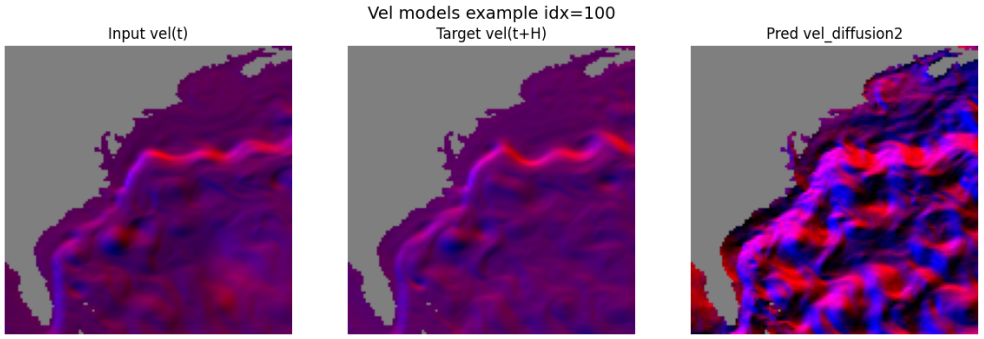


Fig. 21 Diffusion velocity prediction.

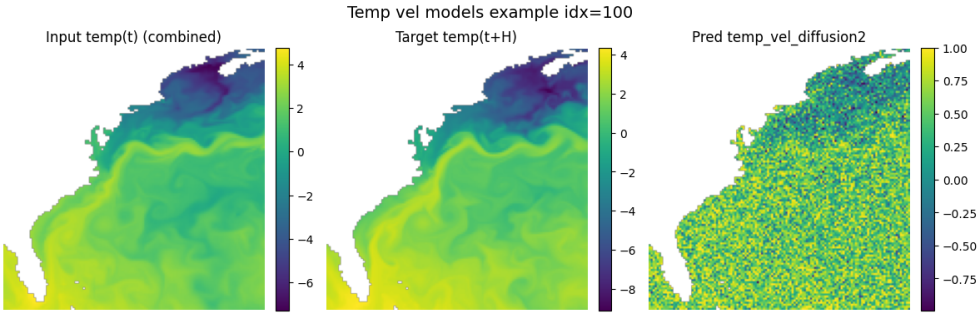


Fig. 22 Diffusion temperature prediction using velocity.

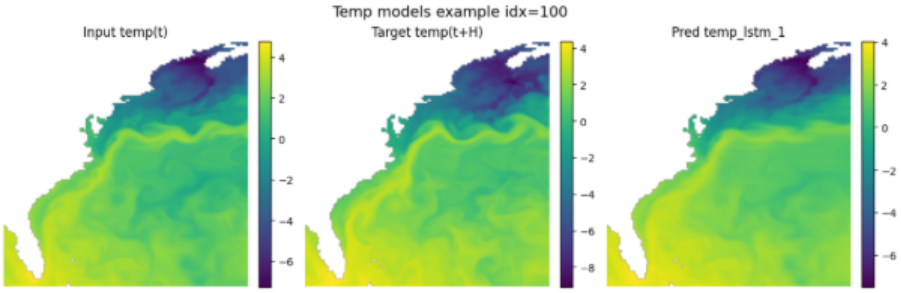


Fig. 23 LSTM temperature prediction.

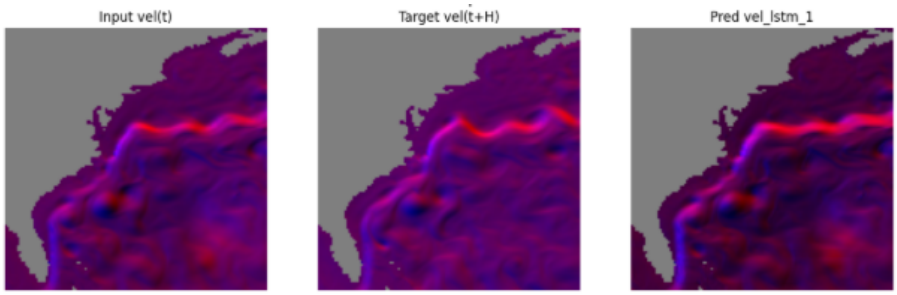


Fig. 24 LSTM velocity prediction.

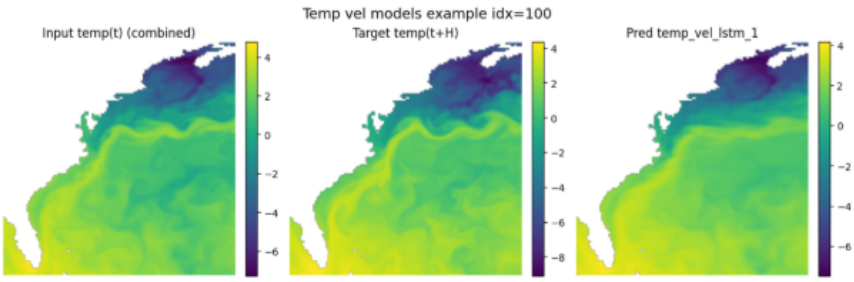


Fig. 25 LSTM remperature prediction using velocity.

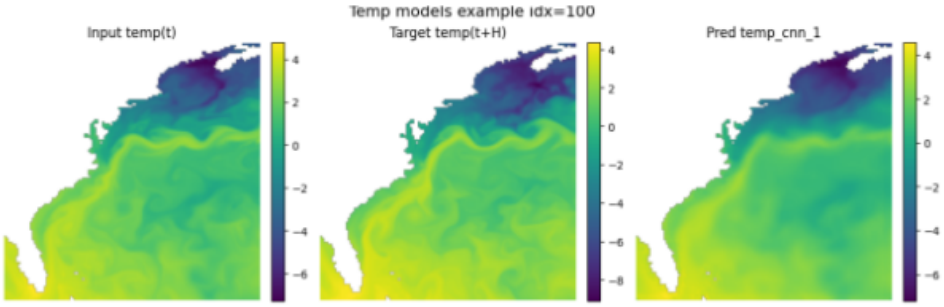


Fig. 26 CNN temperature prediction.

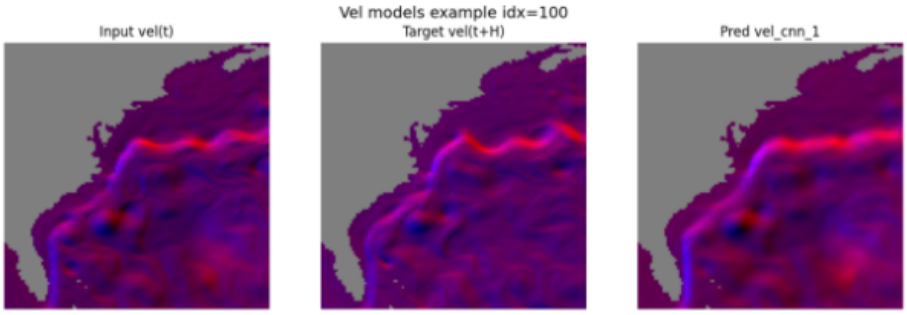


Fig. 27 CNN velocity prediction.

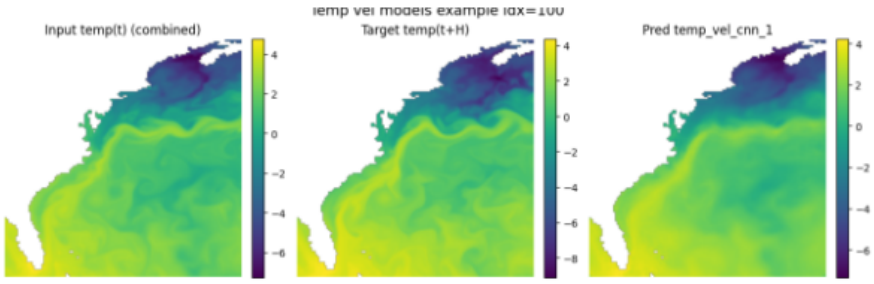


Fig. 28 CNN temperature prediction using velocity.

8 DISCUSSION

The comparison between CNN and LSTM across tasks highlights a consistent pattern: on this spatially gridded, moderately sized dataset, a fully convolutional architecture is more data-efficient and easier to optimize than a recurrent model. The CNN achieves lower RMSE for all three prediction tasks and does so with shorter training time and fewer stability issues during hyperparameter search. In contrast, the LSTM is more sensitive to learning rate, hidden size and number of layers, and exhibits signs of overfitting, especially in the Vel→Vel task. The LSTM exhibits clear optimization instability on the Vel→Vel task. As shown by the training/validation loss curves, the loss decreases overall but presents abrupt spikes and a transient shift to a higher-loss regime before early stopping. This behavior is more consistent with unstable updates (e.g., overshooting) than with smooth overfitting alone, and likely stems from the sensitivity of recurrent models to the learning rate and gradient dynamics under small-batch training. Practical remedies include reducing the learning rate, enabling gradient clipping (e.g., enforcing $\|\nabla\|_2 \leq 1$), and using a learning-rate scheduler (such as cosine decay or ReduceLROnPlateau) to damp sudden parameter jumps. Additional stabilizers such as dropout in the recurrent layers, weight decay, or replacing the LSTM with a gated variant that is easier to optimize (e.g., GRU) may further improve robustness. Finally, increasing the batch size (when feasible) or applying gradient accumulation can reduce gradient noise and help prevent the intermittent divergence observed in this task.

Adding velocity as an input channel to the temperature forecast yields a modest but systematic improvement for the CNN. This suggests that the network is able to exploit information about horizontal advection in addition to temporal persistence. The gain is relatively small, which is consistent with the short 7-day forecast horizon and the strong autocorrelation of sea-surface temperature itself; nevertheless, it indicates that including dynamical variables can help refine deterministic forecasts.

The diffusion model behaves quite differently from the deterministic networks. Training is computationally expensive and requires careful manual tuning. Even with a deeper UNet and a well-chosen linear β schedule, the best diffusion configuration does not match the CNN's and LSTM's RMSE on the Temp→Temp task. However,

diffusion provides two advantages that are not captured by a single RMSE number: it offers a generative description of the forecast distribution, and it naturally produces ensembles whose spread reflects forecast uncertainty.

9 Conclusions

In this project we built an end-to-end pipeline to forecast North Atlantic sea-surface temperature using GLORYS reanalysis data and three distinct deep-learning paradigms: fully convolutional CNNs, recurrent LSTMs, and conditional diffusion models based on a UNet backbone. We formulated three prediction tasks—Temp→Temp, Vel→Vel and Temp+Vel→Temp—and evaluated models on a 7-day forecast horizon over a 10-year period. All models were trained with a land mask and standard normalization, and evaluated using RMSE over ocean points.

The main finding is that, for the current data and model scales, CNNs provide the best balance of accuracy, robustness and computational cost. They outperform LSTMs on all tasks, suffer less from overfitting, and are easier to tune. Incorporating velocity as an additional input channel leads to a small but consistent improvement in temperature forecasts, confirming that horizontal currents carry useful predictive information on weekly timescales.

The diffusion model, while underperforming the CNN in terms of RMSE, show potential for generate realistic temperature fields and celocity fields.

References

[1] J.-M. Lellouche, E. Greiner, R. Bourdallé-Badie, G. Garric, A. Melet, M. Drévillon, C. Bricaud, M. Hamon, O. Le Galloudec, C. Regnier, T. Candela, C.-E. Testut, F. Gasparin, G. Ruggiero, M. Benkiran, Y. Drillet and P.-Y. Le Traon, "The Copernicus Global 1/12° Oceanic and Sea Ice GLORYS12 Reanalysis," *Frontiers in Earth Science*, vol. 9, 698876, 2021.

[2] Copernicus Marine Service, "Global Ocean Physics Reanalysis (GLORYS12V1)," product GLOBAL_MULTIYEAR_PHY_001_030, DOI: 10.48670/moi-00021, 2023. Available at https://data.marine.copernicus.eu/.

[3] J. Ho, A. Jain and P. Abbeel, "Denoising Diffusion Probabilistic Models," in *Advances in Neural Information Processing Systems* 33, pp. 6840–6851, 2020.

[4] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proc. MICCAI 2015*, LNCS 9351, pp. 234–241, 2015.

[5] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] P. von Platen, S. Patil, A. Lozhkov, P. Cuenca, N. Lambert, K. Rasul, M. Davaadorj, D. Nair, S. Paul, W. Berman, Y. Xu, S. Liu and T. Wolf, "Diffusers: State-of-the-Art Diffusion Models," GitHub repository, 2022. Available at https://github.com/huggingface/diffusers.

Appendix: Code Repository

The code and experiments are available on GitHub: https://github.com/Ethreall418/EAEE-4000-Final-Project