Operator Learning with Neural Fields: Tackling PDEs on General Geometries
Supplemental Material

Anonymous Author(s)
Affiliation
Address
email

A Dataset Details

A.1 Initial Value Problem

We use the datasets from Pfaff et al. (2021), and take the first and last frames of each trajectory as the input and output data for the initial value problem.

Cylinder The dataset includes computational fluid dynamics (CFD) simulations of the flow around a cylinder, governed by the incompressible Navier-Stokes equation. These simulations were generated using COMSOL software, employing an irregular 2D-triangular mesh. The trajectory consists of 600 timestamps, with a time interval of $\Delta t = 0.01s$ between each timestamp.

Airfoil The dataset contains CFD simulations of the flow around an airfoil, following the compressible Navier-Stokes equation. These simulations were conducted using SU2 software, using an irregular 2D-triangular mesh. The trajectory encompasses 600 timestamps, with a time interval of $\Delta t = 0.008s$ between each timestamp.

A.2 Dynamics Modeling

2D-Navier-Stokes (*Navier-Stokes*) We consider the 2D Navier-Stokes equation as presented in Li et al. (2021); Yin et al. (2022). This equation models the dynamics of an incompressible fluid on a rectangular domain $\Omega = [-1, 1]^2$. The PDE writes as :

$$\frac{\partial w(x,t)}{\partial t} = -u(x,t)\nabla w(x,t) + \nu \Delta w(x,t) + f, x \in [-1,1]^2, t \in [0,T] \tag{6}$$

$$w(x,t) = \nabla \times u(x,t), x \in [-1,1]^2, t \in [0,T] \tag{7}$$

$$\nabla u(x,t) = 0, x \in [-1,1]^2, t \in [0,T] \tag{8}$$

where u is the velocity, w the vorticity. ν is the fluid viscosity, and f is the forcing term, given by:

$$f(x_1, x_2) = 0.1 \left(\sin(2\pi(x_1 + x_2)) + \cos(2\pi(x_1 + x_2))\right), \forall x \in \Omega \tag{9}$$

For this problem, we consider periodic boundary conditions.

By sampling initial conditions as in Li et al. (2021), we generated different trajectories on a 256×256 regular spatial grid and with a time resolution $\delta t = 1$. We retain the trajectory starting from the 20th timestep so that the dynamics is sufficiently expressed. The final trajectories contains 40 snapshots at time $t = 20, 21, \cdots, 59$. As explained in section 4, we divide these long trajectories into 2 parts : the 20 first frames are used during the training phase and are denoted as *In-t* throughout this paper. The 20 last timesteps are reserved for evaluating the extrapolation capabilities of the models and are the *Out-t* part of the trajectories. In total, we collected 256 trajectories for training, and 16 for evaluation.

3D-Spherical Shallow-Water (*Shallow-Water*). We consider the shallow-water equation on a sphere describing the movements of the Earth's atmosphere:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -f \cdot k \times u - g\nabla h + \nu \Delta u \tag{10}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = -h\nabla \cdot u + \nu \Delta h \tag{11}$$

where $\frac{\mathrm{d}}{\mathrm{d}t}$ is the material derivative, k is the unit vector orthogonal to the spherical surface, u is the velocity field tangent to the surface of the sphere, which can be transformed into the vorticity $w = \nabla \times u$, h is the height of the sphere. We generate the data with the *Dedalus* software (Burns et al., 2020), following the setting described in Yin et al. (2022), where a symmetric phenomena can be seen for both northern and southern hemisphere. The initial zonal velocity u_0 contains two non-null symmetric bands in the both hemispheres, which are parallel to the circles of latitude. At each latitude and longitude $\phi, \theta \in [-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\pi, \pi]$:

$$u_0(\phi,\theta) = \begin{cases} \left(\frac{u_{max}}{e_n} \exp\left(\frac{1}{(\phi-\phi_0)(\phi-\phi_1)}\right), 0\right) & \text{if } \phi \in (\phi_0, \phi_1), \\ \left(\frac{u_{max}}{e_n} \exp\left(\frac{1}{(\phi+\phi_0)(\phi+\phi_1)}\right), 0\right) & \text{if } \phi \in (-\phi_1, -\phi_0), \\ (0,0) & \text{otherwise.} \end{cases} \tag{12}$$

where u_{max} is the maximum velocity, $\phi_0 = \frac{\pi}{7}, \phi_1 = \frac{\pi}{2} - \phi_0$, and $e_n = \exp(-\frac{4}{(\phi_1-\phi_0)^2})$. The water height h_0 is initialized by solving a boundary value conditioned problem as in Galewsky et al. (2004) which is perturbed by adding h_0' to h_0:

$$h_0'(\phi,\theta) = \hat{h}\cos(\phi)\exp\left(-\left(\frac{\theta}{\alpha}\right)^2\right)\left[\exp\left(-\left(\frac{\phi_2-\phi}{\beta}\right)^2\right) + \exp\left(-\left(\frac{\phi_2+\phi}{\beta}\right)^2\right)\right]. \tag{13}$$

where $\phi_2 = \frac{\pi}{4}, \hat{h} = 120\text{m}, \alpha = \frac{1}{3}$ and $\beta = \frac{1}{15}$ are constants defined in Galewsky et al. (2004). We simulated the phenomenon using Dedalus Burns et al. (2020) on a latitude-longitude grid (lat-lon). The original grid size was 128 (lat) $\times$ 256 (lon), which we downsampled to obtain grids of size 64×128. To generate trajectories, we sampled u_{max} from a uniform distribution $\mathcal{U}(60, 80)$. Snapshots were captured every hour over a duration of 320 hours, resulting in trajectories with 320 timestamps. We created 16 trajectories for the training set and 2 trajectories for the test set. However, since the dynamical phenomena in the initial timestamps were less significant, we only considered the last 160 snapshots. Each long trajectory is then sliced into sub-trajectories of 40 timestamps each. As a result, the training set contains 64 trajectories, while the test set contains 8 trajectories. It is worth noting that the data was also scaled to a reasonable range: the height h was scaled by a factor of 3×10^3, and the vorticity w was scaled by a factor of 2.

A.3 Geometric Design

We use the datasets provided by Li et al. (2022a) and adopt the original authors' train/test split for our experiments.

Euler's Equation (*Naca-Euler*). We consider the transonic flow over an airfoil, where the governing equation is Euler equation, as follows:

$$\frac{\partial \rho_f}{\partial t} + \nabla \cdot (\rho_f u) = 0, \frac{\partial \rho_f u}{\partial t} + \nabla \cdot (\rho_f u \otimes u + p\mathbb{I}) = 0, \frac{\partial E}{\partial t} + \nabla \cdot ((E+p)u) = 0, \tag{14}$$

where ρ_f is the fluid density, u is the velocity vector, p is the pressure, and E is the total energy. The viscous effect is ignored. The far-field boundary condition is $\rho_\infty = 1$, $p_\infty = 1.0$, $M_\infty = 0.8$, $AoA = 0$, where M_∞ is the Mach number and AoA is the angle of attack. At the airfoil, a no-penetration condition is imposed. The shape parameterization of the airfoil follows the design element approach. The initial NACA-0012 shape is mapped onto a "cubic" design element with 8 control nodes, and the initial shape is morphed to a different one following the displacement field of the control nodes of the design element. The displacements of control nodes are restricted to the vertical direction only, with prior $d \sim \mathcal{U}[-0.05, 0.05]$.

We have access to 1000 training data and 200 test data, generated with a second-order implicit finite volume solver. The C-grid mesh with about (200×50) quadrilateral elements is used, and the mesh is adapted near the airfoil but not the shock. The mesh point locations and Mach number on these mesh points are used as input and output data.

Hyper-elastic material (*Elasticity*). The governing equation of a solid body can be written as

$$\rho_s \frac{\partial^2 u}{\partial t^2} + \nabla \cdot \sigma = 0$$

where ρ_s is the mass density, u is the displacement vector, and σ is the stress tensor. Constitutive models, which relate the strain tensor ε to the stress tensor, are required to close the system. We consider the unit cell problem $\Omega = [0,1] \times [0,1]$ with an arbitrary shape void at the center, which is depicted in Figure 2(a). The prior of the void radius is $r = 0.2 + 0.2$ with $\tilde{r} \sim \mathcal{N}(0, 42(-\nabla + 32)^{-1})$, $1 + \exp(\tilde{r})$, which embeds the constraint $0.2 \leq r \leq 0.4$. The unit cell is clamped on the bottom edges and tension traction $t = [0, 100]$ is applied on the top edge. The material is the incompressible Rivlin-Saunders material with energy density function parameters $C_1 = 1.863 \times 10^5$ and $C_1 = 9.79 \times 10^3$. The data was generated with a finite element solver with about 100 quadratic quadrilateral elements. The inputs a are given as point clouds with a size around 1000. The target output is stress.

Navier-Stokes Equation (*Pipe*). We consider the incompressible flow in a pipe, where the governing equation is the incompressible Navier-Stokes equation, as following,

$$\frac{\partial v}{\partial t} + (v \cdot \nabla)v = -\nabla p + \mu \nabla^2 v, \quad \nabla \cdot v = 0$$

where v is the velocity vector, p is the pressure, and $\mu = 0.005$ is the viscosity. The parabolic velocity profile with maximum velocity $v = [1, 0]$ is imposed at the inlet. A free boundary condition is imposed at the outlet, and a no-slip boundary condition is imposed at the pipe surface. The pipe has a length of 10 and width of 1. The centerline of the pipe is parameterized by 4 piecewise cubic polynomials, which are determined by the vertical positions and slopes on 5 spatially uniform control nodes. The vertical position at these control nodes obeys $d \sim \mathcal{U}[-2, 2]$, and the slope at these control nodes obeys $d \sim \mathcal{U}[-1, 1]$.

We have access to 1000 training data and 200 test data, generated with an implicit finite element solver using about 4000 Taylor-Hood Q2-Q1 mixed elements. The mesh point locations (129×129) and horizontal velocity on these mesh points are used as input and output data.

B Implementation Details

We implemented all experiments with PyTorch (Paszke et al., 2019). The code is available at `https://anonymous.4open.science/r/coral-0348/`. We estimate the computation time needed for development and the different experiments to approximately 400 days.

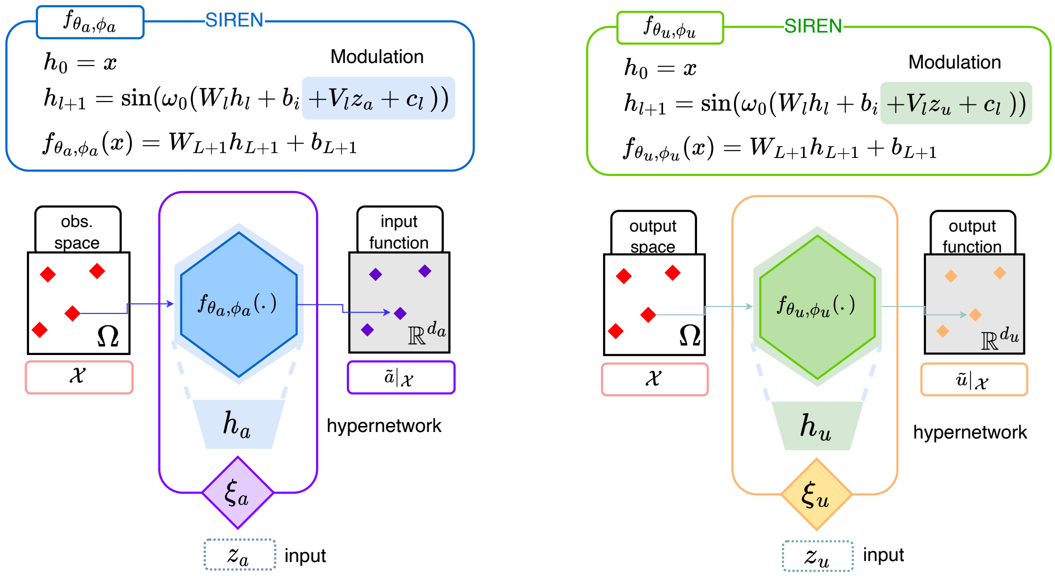
B.1 CORAL

B.1.1 Architecture Details

SIREN initialization. We use for SIREN the same initialization scheme as in Sitzmann et al. (2020b), i.e., sampling the weights of the first layer according to a uniform distribution $\mathcal{U}(-1/d, 1/d)$ and the next layers according to $\mathcal{U}(-\frac{1}{w_0}\sqrt{\frac{6}{d_{in}}}, \frac{1}{w_0}\sqrt{\frac{6}{d_{in}}})$. We use the default PyTorch initialization for the hypernetwork.

Decode with shift-modulated SIREN. Initially, we attempted to modulate both the scale and shift of the activation, following the approach described in Perez et al. (2018). However, we did not observe any performance improvement by employing both modulations simultaneously. Consequently, we decided to focus solely on shift modulations, as it led to a more stable training process and reduced the size of the modulation space by half. We provide an overview of the decoder with the shift-modulated SIREN in Figure 3.

Encode with *auto-decoder*. We provide a schematic view of the input encoder in Figure 4. The *auto-decoding* process starts from a code $z_a = 0$ and performs K steps of gradient descent over this latent code to minimize the reconstruction loss.



(a) The hypernetwork h_a maps the input code z_a to the modulations ϕ_a. The modulations shift the activations at each layer of the SIREN.

(b) The hypernetwork h_u maps the input code z_u to the modulations ϕ_u. The modulations shift the activations at each layer of the SIREN.

Figure 3: Architecture of the input and output decoders ξ_a, ξ_u. They can be queried on any coordinate $x \in \Omega$. We use the same notation for both, even though the parameters are different.

Process with MLP. We use an MLP with skip connections and Swish activation functions. Its forward function writes $g_\psi(z) = \text{Block}_k \circ ... \circ \text{Block}_1(z)$, where Block is a two-layer MLP with skip connections:

$$\text{Block}(z) = z + \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot z + \mathbf{b}_1) + \mathbf{b}_2) \tag{15}$$

In Equation (15), σ denotes the feature-wise Swish activation. We use the version with learnable parameter β; $\sigma(z) = z \cdot \text{sigmoid}(\beta z)$.

B.1.2 Training Details

The training is done in two steps. First, we train the modulated INRs to represent the data. We show the details with the pseudo-code in Algorithms 1 and 2. α is the inner-loop learning rate while λ is the outer loop learning rate, which adjusts the weights of the INR and hypernetwork. Then, once the INRs have been fitted, we obtain the latent representations of the training data, and use these latent codes to train the forecast model g_ψ (See Algorithm 3). We note λ_ψ the learning rate of g_ψ.

Z-score normalization. As the data is encoded using only a few steps of gradients, the resulting standard deviation of the codes is very small, falling within the range of [1e-3, 5e-2]. However, these "raw" latent representations are not suitable as-is for further processing. To address this, we normalize the codes by subtracting the mean and dividing by the standard deviation, yielding the normalized code: $z_{\text{norm}} = \frac{z - \text{mean}}{\text{std}}$. Depending on the task, we employ slightly different types of normalization:

1. Initial value problem: • *Cylinder*: We normalize the inputs and outputs code with the same mean and standard deviation. We compute the statistics feature-wise, across the inputs and outputs. • *Airfoil*: We normalize the inputs and outputs code with their respective mean and standard deviation. The statistics are real values.
2. Dynamics modeling: We normalize the codes with the same mean and standard deviation. The statistics are computed feature-wise, over all training trajectories and all available timestamps (i.e. over *In-t*).
3. Geometric design: We normalize the input codes only, with feature-wise statistics.

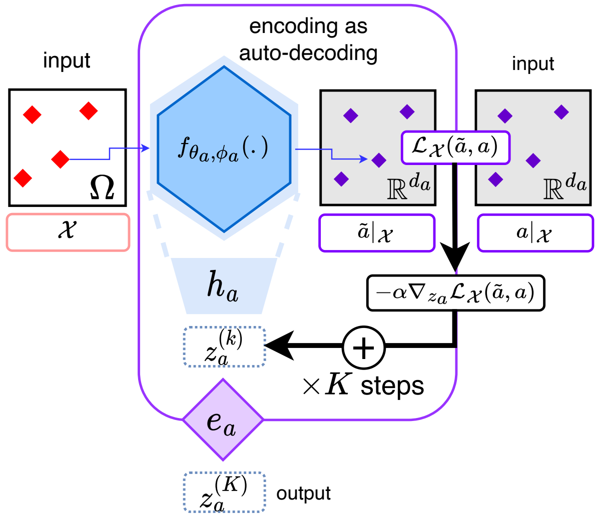


Figure 4: Starting from a code $z_a^{(0)} = 0$, the input encoder e_a performs K inner steps of gradient descent over z_a to minimize the reconstruction loss $\mathcal{L}_{\mathcal{X}}(\tilde{a}, a)$ and outputs the resulting code $z_a^{(K)}$ of this optimization process. During training, we accumulate the gradients of this encoding phase and back-propagate through the K inner-steps to update the parameters θ_a and w_a. At inference, we encode new inputs with the same number of steps K and the same learning rate α, unless stated otherwise. The output encoder works in the same way during training, and is not used at inference.

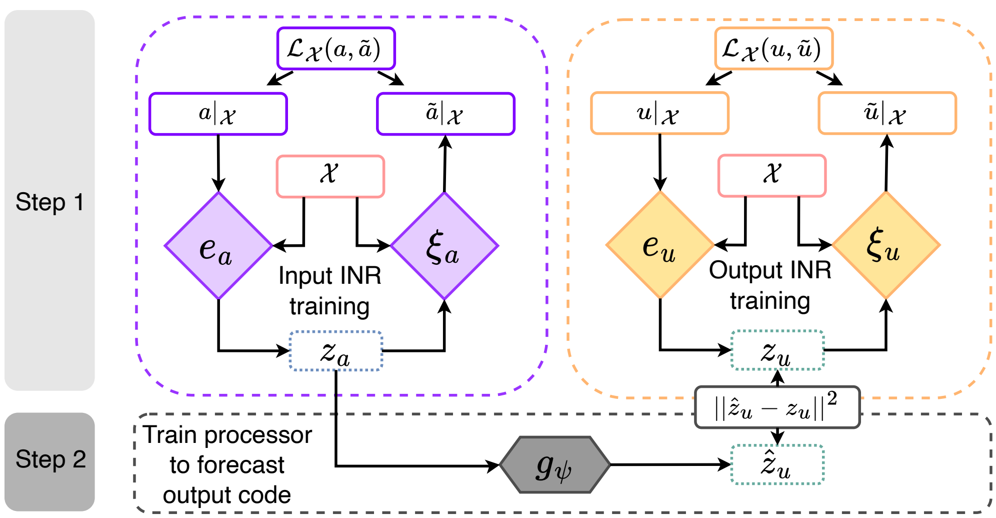


Figure 5: Proposed training for CORAL. (1) We first learn to represent the data with the input and output INRs. (2) Once the INRs are trained, we obtain the latent represenations and fix the pairs of input and output codes (z_{a_i}, z_{u_i}). We then train the processor to minimize the distance between the processed code $g_\psi(z_{a_i})$ and the output code z_{u_i}.

B.1.3 Inference Details

We present the inference procedure in Algorithm 4. It is important to note that the input and output INRs, f_{θ_a} and f_{θ_u}, respectively, accept the "raw" codes as inputs, whereas the processor expects a normalized latent code. Therefore, after the encoding steps, we normalize the input code. Additionally, we may need to denormalize the code immediately after the processing stage. It is worth mentioning that we maintain the same number of inner steps as used during training, which is 3 for all tasks.

Algorithm 1: Training of the input INR

while no convergence **do**
  Sample batch $\mathcal{B}$ of data $(a_i)_{i\in\mathcal{B}}$;
  Set codes to zero $z_{a_i} \leftarrow 0, \forall i \in \mathcal{B}$;
  for $i \in \mathcal{B}$ and step $\in \{1, ..., K_a\}$ **do**
    $z_{a_i} \leftarrow z_{a_i} - \alpha_a \nabla_{z_{a_i}} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_a, h_a(z_{a_i})}, a_i)$; // input encoding inner step
  end
  /* outer loop update */
  $\theta_a \leftarrow \theta_a - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{\theta_a} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_a, h_a(z_{a_i})}, a_i)$;
  $w_a \leftarrow w_a - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{w_a} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_a, h_a(z_{a_i})}, a_i)$
end

Algorithm 2: Training of the output INR

while no convergence **do**
  Sample batch $\mathcal{B}$ of data $(a_i, u_i)_{i\in\mathcal{B}}$;
  Set codes to zero $z_{u_i} \leftarrow 0, \forall i \in \mathcal{B}$;
  for $i \in \mathcal{B}$ and step $\in \{1, ..., K_u\}$ **do**
    $z_{u_i} \leftarrow z_{u_i} - \alpha_u \nabla_{z_{u_i}} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u, h_u(z_{u_i})}, u_i)$; // output encoding inner step
  end
  /* outer loop update */
  $\theta_u \leftarrow \theta_u - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{\theta_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u, h_u(z_{u_i})}, u_i)$;
  $w_u \leftarrow w_u - \lambda \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_{w_u} \mathcal{L}_{\mathcal{X}_i}(f_{\theta_u, h_u(z_{u_i})}, u_i)$
end

Algorithm 3: Training of the processor

while no convergence **do**
  Sample batch $\mathcal{B}$ of codes $(z_{a_i}, z_{u_i})_{i\in\mathcal{B}}$;
  /* processor update */
  $\psi \leftarrow \psi - \lambda_\psi \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \nabla_\psi \mathcal{L}(g_\psi(z_{a_i}), z_{u_i})$;
end

Algorithm 4: CORAL Inference, given a function a

Set code to zero $z_a \leftarrow 0$;
for step $\in \{1, ..., K_a\}$ **do**
  $z_a \leftarrow z_a - \alpha_a \nabla_{z_a} \mathcal{L}_{\mathcal{X}}(f_{\theta_a, h_a(z_a)}, a)$; // input encoding inner step
end
$\hat{z}_u = g_\psi(z_a)$; // process latent code
$\hat{u} = f_{\theta_u, h_u(\tilde{z}_u)}$; // decode output function

B.1.4 Choice of Hyperparameters

We recall that d_z denotes the size of the code, w_0 is a hyperparameter that controls the frequency bandwith of the SIREN network, λ is the outer-loop learning rate (on $f_{\theta,\phi}$ and h_w), α is the inner-loop learning rate, K is the number of inner steps used during training and encoding steps at test time, λ_ψ is the learning rate of the MLP or NODE. In some experiments we learn the inner-loop learning rate α, as in Li et al. (2017). In such case, the meta-α learning rate is an additional parameter that controls how fast we move α from its initial value during training. When not mentioned we simply report α in the tables below, and otherwise we report the initial learning rate and this meta-learning-rate.

We use the Adam optimizer during both steps of the training. For the training of the Inference / Dynamics model, we use a learning rate scheduler which reduces the learning rate when the loss has stopped improving. The threshold is set to 0.01 in the default relative threshold model in PyTorch, with a patience of 250 epochs w.r.t. the train loss. The minimum learning rate is 1e-5.

Initial Value Problem We provide the list of hyperparameters used for the experiments on *Cylinder* and *Airfoil* in Table 4.

Dynamics Modeling Table 5 summarizes the hyperparameters used in our experiments for dynamics modeling on datasets *Navier-Stokes* and *Shallow-Water* (Table 2).

Furthermore, to facilitate the training of the dynamics within the NODE, we employ Scheduled Sampling, following the approach described in Bengio et al. (2015). At each timestep, there is a

Table 4: **CORAL hyper-parameters for IVP/ Geometric design**

	Hyper-parameter	*Cylinder*	*Airfoil*	*NACA-Euler*	*Elasticity*	*Pipe*
f_{θ_a,ϕ_a} / f_{θ_u,ϕ_u}	d_z	128	128	128	128	128
	depth	4	5	4	4	5
	width	256	256	256	256	128
	ω_0	30	30 / 50	5 / 15	10 / 15	5 / 10
SIREN Optimization	batch size	32	16	32	64	16
	epochs	2000	1500	5000	5000	5000
	λ	5e-6	5e-6	1e-4	1e-4	5e-5
	α	1e-2	1e-2	1e-2	1e-2	1e-2
	meta-α learning rate	0	5e-6	1e-4	1e-4	5e-5
	K_a / K_u	3	3	3	3	3
g_ψ	depth	3	3	3	3	3
	width	64	64	64	64	128
	activation	Swish	Swish	Swish	Swish	Swish
Inference Optimization	batch size	32	16	64	64	64
	epochs	2000	100	10000	10000	10000
	λ_ψ	1e-3	1e-3	1e-3	1e-3	1e-3
	Scheduler decay	0	0	0.9	0.9	0.9

Table 5: **CORAL hyper-parameters for dynamics modeling**

	Hyper-parameter	*Navier-Stokes*	*Shallow-Water*
INR	d_z	128	256
	depth	4	6
	width	128	256
	ω_0	10	10
INR Optimization	batch size	64	16
	epochs	10, 000	10, 000
	λ	5e-6	5e-6
	α	1e-2	1e-2
	K	3	3
NODE	depth	3	3
	width	512	512
	activation	Swish	Swish
	solver	RK4	RK4
Dynamics Optimization	batch size	32	16
	epochs	10, 000	10, 000
	λ_ψ	1e-3	1e-3
	Scheduler decay	0.75	0.75

probability of $\epsilon\%$ for the integration of the dynamics through the ODE solver to be restarted using the training snapshots. This probability gradually decreases during the training process. Initially, we set $\epsilon_{\text{init}} = 0.99$, and every 10 epochs, we multiply it by 0.99. Consequently, by the end of the training procedure, the entire trajectory is computed with the initial condition.

Geometric Design We provide the list of hyperparameters used for the experiments on *NACA-Euler*, *Elasticity*, and *Pipe* in Table 4.

B.2 Baseline Implementation

We detail in this section the architecture and hyperparameters used for the training of the baselines presented in Section 4.

Initial Value Problem We use the following baselines for the Initial Value Problem task.

- **NodeMLP**. We use a ReLU-MLP with 3 layers and 512 neurons. We train it for 10000 epochs. We use a learning rate of 1e-3 and a batch size of 64.
- **GraphSAGE**. We use the implementation from torch-geometric (Fey & Lenssen, 2019), with 6 layers of 64 neurons. We use ReLU activation. We train the model for 400 epochs for *Airfoil* and 4,000 epochs for *Cylinder*. We build the graph using the 16 closest nodes. We use a learning rate of 1e-3 and a batch size of 64.
- **MP-PDE**: We implement MP-PDE as a 1-step solver, where the time-bundling and pushforward trick do not apply. We use 6 message-passing blocks and 64 hidden features. We build the graph with the 16 closest nodes. We use a learning rate of 1e-3 and a batch size of 16. We train for 500 epochs on *Airfoil* and 1000 epochs on *Cylinder*.

Dynamics Modeling All our baselines are implemented in an auto-regressive (AR) manner to perform forecasting.

- **DeepONet**: We use a DeepONet in which both Branch Net and Trunk Net are 4-layers MLP with 100 neurons. The model is trained for $10,000$ epochs with a learning rate of 1e-5. To complete the upsampling studies, we used a modified DeepONet forward which computes as follows: (1) Firstly, we compute an AR pass on the training grid to obtain a prediction of the complete trajectory with the model on the training grid. (2) We use these prediction as input of the branch net for a second pass on the up-sampling grid to obtain the final prediction on the new grid.
- **FNO**: FNO is trained for 2000 epochs with a learning rate of 1e-3. We used 12 modes and a width of 32 and 4 Fourier layers. We also use a step scheduler every 100 epochs with a decay of 0.5.
- **MP-PDE**: We implement MP-PDE with a time window of 1 so that is becomes AR. The MP-PDE solver is composed of a 6 message-passing blocks with 128 hidden features. To build the graphs, we limit the number of neighbors to 8. The optimization was performed on 10000 epochs with a learning rate of 1e-3 and a step scheduler every 2000 epochs until 10000. We decay the learning rate of 0.4 with weight decay 1e-8.
- **DINo**: DINo uses MFN model with respectively width and depth of 64 and 3 for Navier-Stokes (NS), and 256 and 6 for Shallow-Water (SW). The encoder proceeds to 300 (NS) or 500 (SW) steps to optimize the codes whose size is set to 100 (NS) or 200 (SW). The dynamic is solved with a NODE that uses 4-layers MLP and a hidden dimension of 512 (NS) or 800 (SW). This model is trained for 10000 epochs with a learning rate of 5e-3. We use the same scheduled sampling as for the CORAL training (see appendix B.1.4).

Geometric Design Except for **FactorizedFNO** on *Pipe*, the numbers for **GeoFNO**, **FNO**, **UNet** are taken from Li et al. (2022a) and the numbers for FactorizedFNO are taken from Tran et al. (2023). In the latter we take the 12-layer version which has a comparable model size. We train the 12-layer Factorized FNO on *Pipe* with AdamW for 200 epochs with modes (32, 16), a width of 64, a learning rate of 1e-3 and a weight decay of 1e-4.

C Supplementary Results for Dynamics Modeling

C.1 Robustness to Resolution Changes

We present in Tables 6 and 7 the up-sampling capabilities of CORAL and relevant baselines both *In-t* and *Out-t*, respectively for Navier-Stokes and Shallow-Water.

Table 6: **Up-sampling capabilities** - Test results on Navier-Stokes dataset. Metrics in MSE.

	dataset $\rightarrow$	Navier-Stokes							
$\mathcal{X}_{tr} \downarrow$	$\mathcal{X}_{tr} \rightarrow$	64×64							
	$\mathcal{X}_{te} \rightarrow$	$\mathcal{X}_{tr}$		64×64		128×128		256×256	
		In-t	Out-t	In-t	Out-t	In-t	Out-t	In-t	Out-t
$\pi_{tr} = 100\%$ regular grid	DeepONet	1.47e-2	7.90e-2	1.47e-2	7.90e-2	1.82e-1	7.90e-2	1.82e-2	7.90e-2
	FNO	7.97e-3	1.77e-2	7.97e-3	1.77e-2	8.04e-3	1.80e-2	1.81e-2	7.90e-2
	MP-PDE	5.98e-4	2.80e-3	5.98e-4	2.80e-3	2.36e-2	4.61e-2	4.26e-2	9.77e-2
	DINo	1.25e-3	1.13e-2	1.25e-3	1.13e-2	1.25e-3	1.13e-2	1.26e-3	1.13e-2
	CORAL	**2.02e-4**	**1.07e-3**	**2.02e-4**	**1.07e-3**	**2.08e-4**	**1.06e-3**	**2.19e-4**	**1.07e-3**
$\pi_{tr} = 20\%$ irregular grid	DeepONet	8.35e-1	7.74e-1	8.28e-1	7.74e-1	8.32e-1	7.74e-1	8.28e-1	7.73e-1
	MP-PDE	2.36e-2	1.11e-1	7.42e-2	2.13e-1	1.18e-1	2.95e-1	1.37e-1	3.39e-1
	DINo	**1.30e-3**	9.58e-3	**1.30e-3**	9.59e-3	**1.31e-3**	9.63e-3	**1.32-3**	9.65e-3
	CORAL	1.73e-3	**5.61e-3**	1.55e-3	**4.34e-3**	1.61e-3	**4.38e-3**	1.65e-3	**4.41e-3**
$\pi_{tr} = 5\%$ irregular grid	DeepONet	7.12e-1	7.16e-1	7.22e-1	7.26e-1	7.24e-1	7.28e-1	7.26e-1	7.30e-1
	MP-PDE	1.25e-1	2.92e-1	4.83e-1	1.08	6.11e-1	1.07	6.49e-1	1.08
	DINo	8.21e-2	1.03e-1	7.73e-2	7.49e-2	7.87e-2	7.63e-2	7.96e-2	7.73e-2
	CORAL	**1.56e-2**	**3.65e-2**	**4.19e-3**	**1.12e-2**	**4.30e-3**	**1.14e-2**	**4.37e-3**	**1.14e-2**

Table 7: **Up-sampling capabilities** - Test results on Shallow-water dataset. Metrics in MSE.

	dataset $\rightarrow$	Shallow-water							
$\mathcal{X}_{tr} \downarrow$	$\mathcal{X}_{tr} \rightarrow$	64×128							
	$\mathcal{X}_{te} \rightarrow$	$\mathcal{X}_{tr}$		32×64		64×128		128×256	
		In-t	Out-t	In-t	Out-t	In-t	Out-t	In-t	Out-t
$\pi_{tr} = 100\%$ regular grid	DeepONet	7.07e-3	9.02e-3	1.18e-2	1.66e-2	7.07e-3	9.02e-3	1.18e-2	1.66e-2
	FNO	6.75e-5	**1.49e-4**	7.54e-5	**1.78e-4**	6.75e-5	**1.49e-4**	6.91e-5	**1.52e-4**
	MP-PDE	2.66e-5	4.35e-4	4.80e-2	1.42e-2	2.66e-5	4.35e-4	4.73e-3	1.73e-3
	DINo	4.12e-5	2.91e-3	5.77e-5	2.55e-3	4.12e-5	2.91e-3	6.04e-5	2.58e-3
	CORAL	**3.52e-6**	4.99e-4	**1.86e-5**	5.32e-4	**3.52e-6**	4.99e-4	**4.96e-6**	4.99e-4
irregular grid $\pi_{tr} = 20\%$	DeepONet	1.08e-2	1.10e-2	2.49e-2	3.25e-2	2.49e-2	3.25e-2	2.49e-2	3.22e-2
	MP-PDE	4.54e-3	1.48e-2	4.08e-3	1.30e-2	5.46e-3	1.74e-2	4.98e-3	1.43e-2
	DINo	2.32e-3	5.18e-3	2.22e-3	4.80e-3	2.16e-3	4.64e-3	2.16e-3	4.64e-3
	CORAL	**1.36e-3**	**2.17e-3**	**1.24e-3**	**1.95e-3**	**1.21e-3**	**1.95e-3**	**1.21e-3**	**1.95e-3**
irregular grid $\pi_{tr} = 5\%$	DeepONet	1.02e-2	**1.01e-2**	1.57e-2	1.93e-2	1.57e-2	1.93e-2	1.57e-2	1.93e-2
	MP-PDE	**5.36e-3**	1.81e-2	**5.53e-3**	1.80e-2	**4.33e-3**	1.32e-2	**5.48e-3**	1.74e-2
	DINo	1.25e-2	1.51e-2	1.39e-2	1.54e-2	1.39e-2	1.54e-2	1.39e-2	1.54e-2
	CORAL	8.40e-3	1.25e-2	9.27e-3	**1.15e-2**	9.26e-3	**1.16e-2**	9.26e-3	**1.16e-2**

These tables show that CORAL remains competitive and robust on up-sampled inputs. Other baselines can also predict on denser grids, except for MP-PDE, which over-fitted the training grid.

C.2 Learning a Dynamics on Different Grids

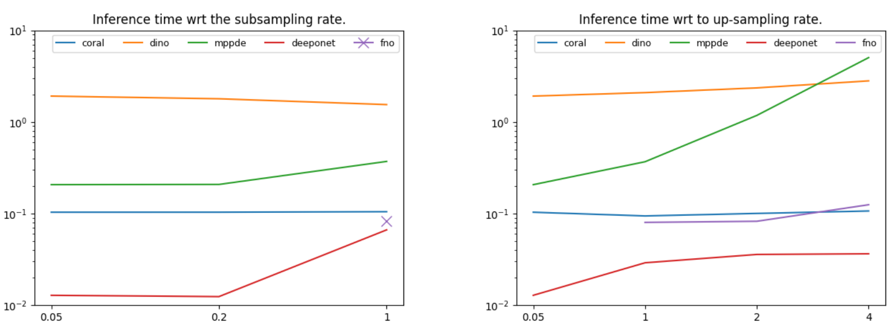
To extend our work, we propose to study how robust is CORAL to changes in grids. In our classical setting, we keep the same grid for all trajectories in the training set and evaluate it on a new grid for the test set. Instead, here, both in train and test sets, each trajectory i has its own grid $\mathcal{X}_i$. Thus, we evaluate CORAL's capability to generalize to grids. We present the results in Table 8. Overall, coordinate-based methods generalize better over grids compared to operator based and discrete methods like DeepONet and MP-PDE which show better or equivalent performance when trained only on one grid. CORAL's performance is increased when trained on different grids; one possible reason is that CORAL overfits the training grid used for all trajectories in our classical setting.

Table 8: **Learning dynamics on different grids** - Test results in the extrapolation setting. Metrics in MSE.

$\mathcal{X}_{tr} \downarrow \mathcal{X}_{te}$	dataset $\rightarrow$	*Navier-Stokes*		*Shallow-Water*	
		In-t	*Out-t*	*In-t*	*Out-t*
$\pi = 20\%$ irregular grid	DeepONet	5.22E−1	5.00E−1	1.11E−2	1.12E−2
	MP-PDE	6.11E−1	6.10E−1	6.80E−3	1.87E−2
	DINo	1.30E−3	1.01E−2	4.12E−4	3.05E−3
	CORAL	**3.21E−4**	**3.03E−3**	**1.15E−4**	**7.75E−4**
$\pi = 5\%$ irregular grid	DeepONet	4.11E−1	4.38E−1	1.11E−2	1.12E−2
	MP-PDE	8.15E−1	1.10	1.22E−2	4.29E−2
	DINo	1.26E−3	1.04E−2	3.89E−3	7.41E−3
	CORAL	**9.82E−4**	**9.71E−3**	**2.22e-3**	**4.89e-3**

C.3 Inference Time

In this section, we evaluate the inference time of CORAL and other baselines w.r.t. the input grid size. We study the impact of the training grid size (different models trained with 5%, 20% and 100% of the grid) (Figure 6a) and the time needed for a model trained (5%) on a given grid to make computation on finer grid size resolution (evaluation grid size) (Figure 6b).



(a) Inference time w.r.t. training grid size (different models).

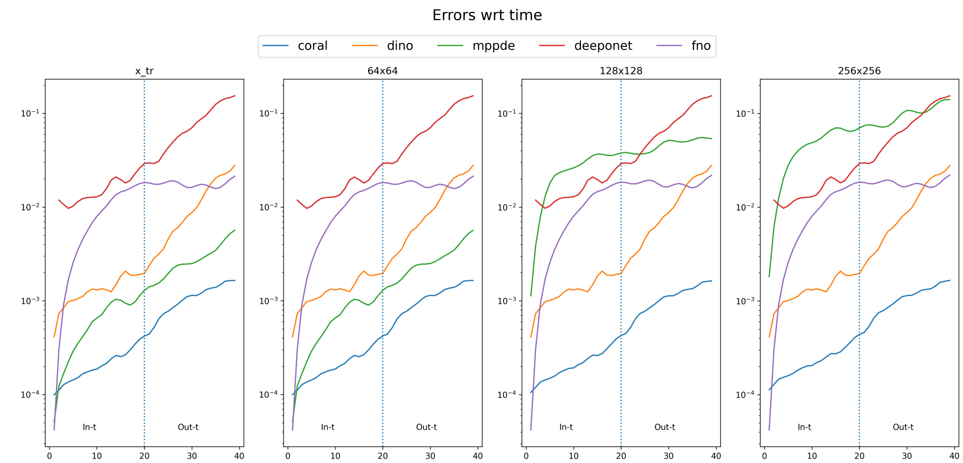
(b) Inference time w.r.t. evaluation grid size (same models).

Figure 6

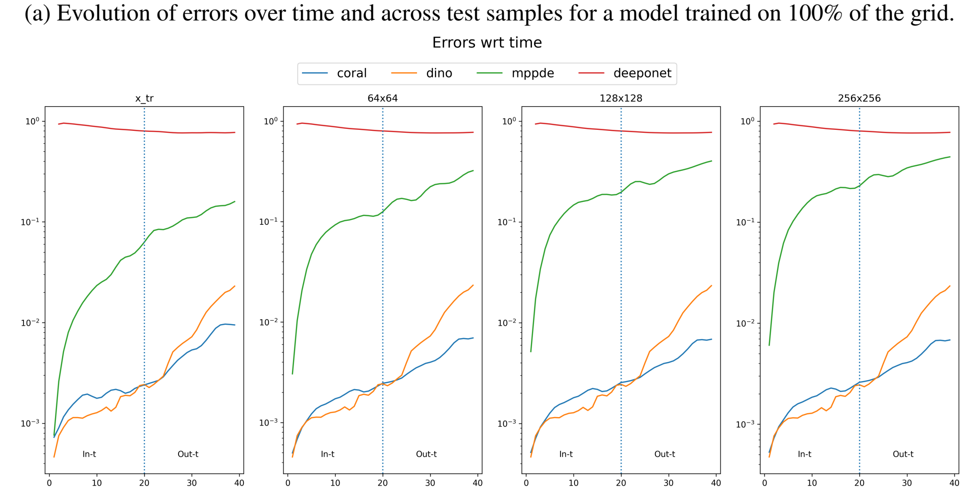
On the graphs presented in Figure 6, we observe that except for the operator baselines, CORAL is also competitive in terms of inference time. MP-PDE inference time increases strongly when inference grid gets denser. The DINo model, which is the only to propose the same properties as CORAL, is much slower when both inference and training grid size evolve. This difference is mainly explained by the number of steps needed to optimize DINo codes. Indeed, DINo requires 100 times more steps than CORAL to compute its code at inference time. Moreover INR-based model's inference time are scaling very well when the input grid increases.

C.4 Propagation of Errors Through Time

In Figures 7a to 7c, we show the evolution of errors as the extrapolation horizon evolves. First, we observe that all baselines propagate error through time, since the trajectories are computed using an auto-regressive approach. Except for the 100%, DeepONet had difficulties to handle the dynamic. It has on all settings the highest error. Then, we observe that for MP-PDE and FNO, the error increases quickly at the beginning of the trajectories. This means that these two models are rapidly propagating error. Finally, both DINo and CORAL have slower increase of the error during *In-t* and *Out-t* periods. However, we clearly see on the graphs that DINo has more difficulties than CORAL to



(a) Evolution of errors over time and across test samples for a model trained on 100% of the grid.



(b) Evolution of errors over time and across test samples for a model trained on 20% of the grid.

(c) Evolution of errors over time and across test samples for a model trained on 5% of the grid.

Figure 7: Errors along a given trajectory.

make predictions out-range. Indeed, while CORAL's error augmentation remains constant as long as the time evolves, DINo has a clear increase.

C.5 Benchmarking INRs for CORAL

We provide some additional experiments for dynamics modeling with CORAL, but with diffrents INRs: MFN (Fathony et al., 2021), BACON (Lindell et al., 2022) and FourierFeatures (Tancik et al., 2020). Experiments have been done on Navier-Stokes on irregular grids sampled from grids of size 128×128. All training trajectories share the same grid and are evaluated on a new grid for test trajectories. Results are reported in Table 9. Note that we used the same learning hyper-parameters for the baselines than those used for SIREN in CORAL. SIREN seems to produce the best codes for dynamics modeling, both for in-range and out-range prediction.

Table 9: **CORAL results with different INRs.** - Test results in the extrapolation setting on *Navier-Stokes* dataset. Metrics in MSE.

$\mathcal{X}_{tr} \downarrow \mathcal{X}_{te}$	INR	*In-t*	*Out-t*
	SIREN	**5.76e-4**	**2.57e-3**
$\pi = 20\%$	MFN	2.21e-3	5.17e-3
irregular grid	BACON	2.90e-2	3.32e-2
	FourierFeatures	1.70e-3	5.67e-3
	SIREN	**1.81e-3**	**4.15e-3**
$\pi = 5\%$	MFN	9.97e-1	9.58e-1
irregular grid	BACON	1.06	8.06e-1
	FourierFeatures	3.60e-1	3.62e-1

D Supplementary Results for Geometric Design

D.1 Inverse Design for NACA-airfoil

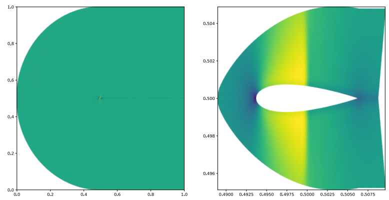
Once trained on *NACA-Euler*, CORAL can be used for the inverse design of a NACA airfoil. We consider an airfoil's shape parameterized by seven spline nodes and wish to minimize drag and maximize lift. We optimize the design parameters in an end-to-end manner. The spline nodes create the input mesh, which CORAL maps to the output velocity field. This velocity field is integrated to compute the drag and the lift, and the loss objective is the squared drag over lift ratio. As can be seen in Figure 8, iterative optimization results in an asymmetric airfoil shape, enhancing progressively the lift coefficient in line with physical expectations. At the end of the optimization we reach a drag value of 0.042 and lift value of 0.322.

E Qualitative results

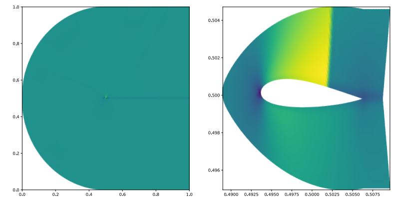
In this section, we show different visualization of the predictions made by CORAL on the three considered tasks in this paper.

E.1 Initial Value Problem

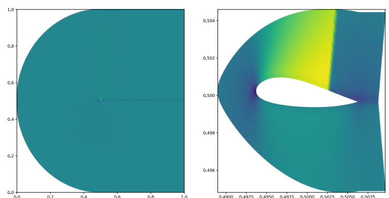
We provide in Figure 9 and Figure 10 visualizations of the inferred values of CORAL on *Cylinder* and *Airfoil*.



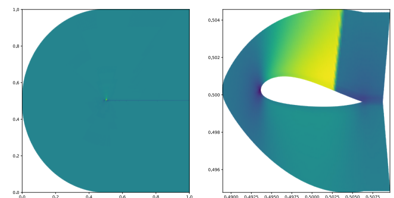
(a) Step = 0



(b) Step = 1000



(c) Step = 3000



(d) Step = 5000

Figure 8: Design optimization of a NACA-Airfoil.

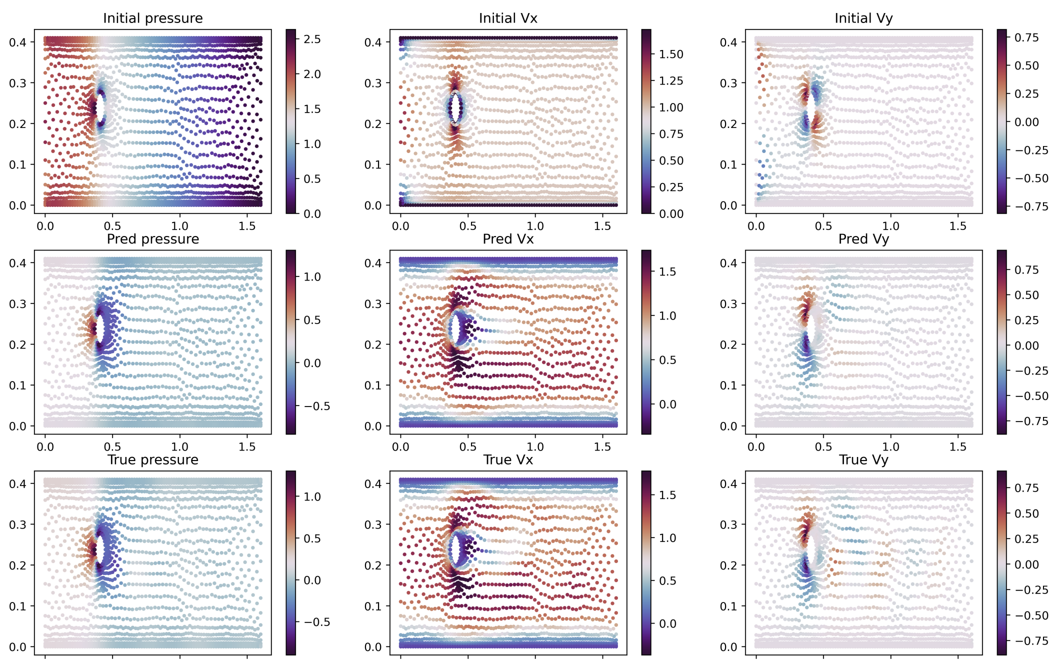


Figure 9: CORAL prediction on *Cylinder*

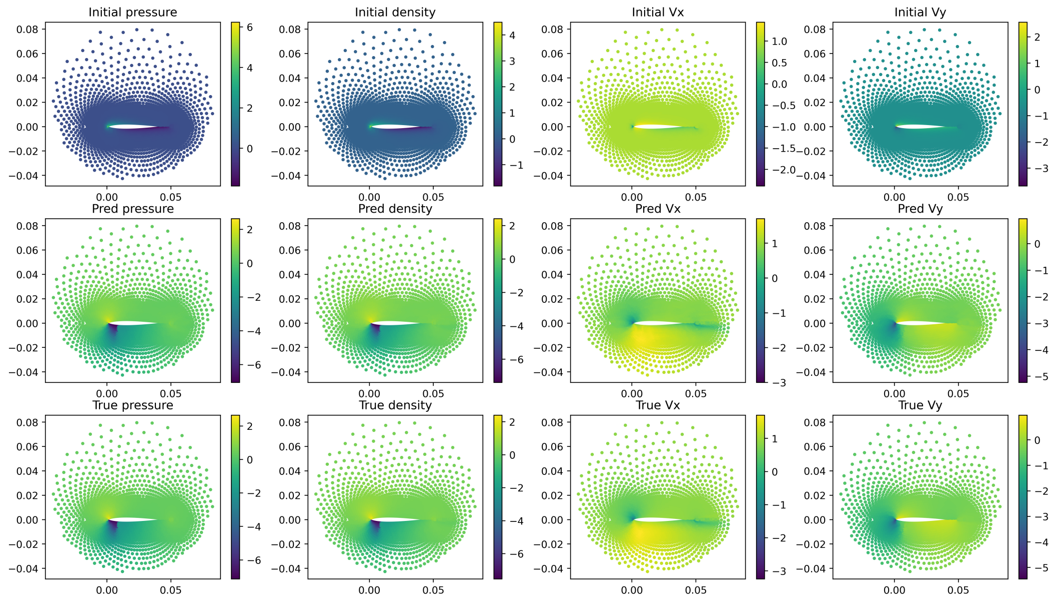


Figure 10: CORAL prediction on *Airfoil*

E.2 Dynamics modeling

We provide in Figure 12 and Figure 11 visualization of the predicted trajectories of CORAL on *Navier-Stokes* and *Shallow-Water*.

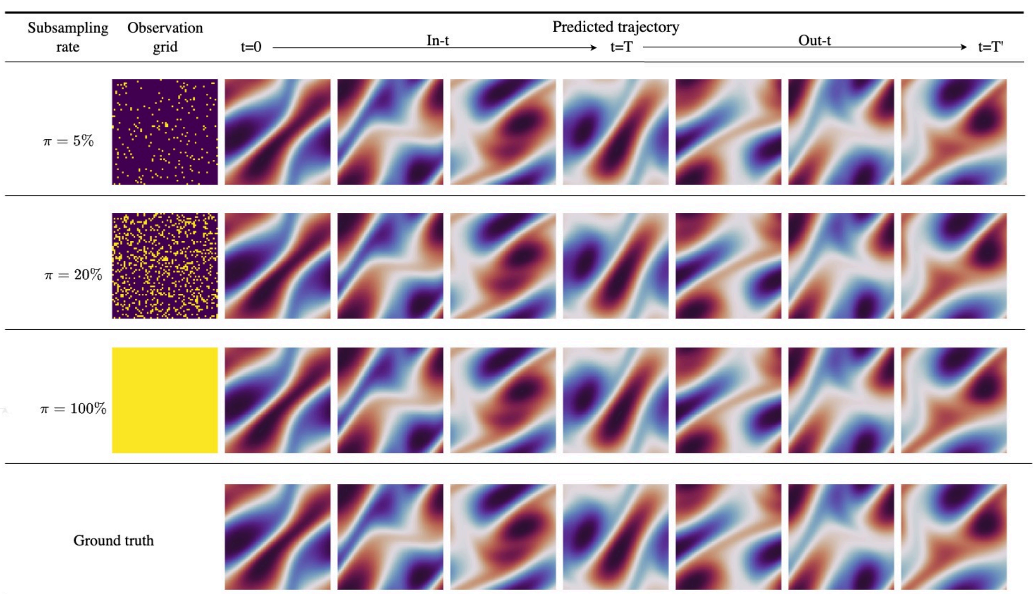


Figure 11: Prediction MSE per frame for CORAL on *Navier-Stokes* with its corresponding training grid $\mathcal{X}$. Each row corresponds to a different sampling rate and the last row is the ground truth. The predicted trajectory is predicted from $t = 0$ to $t = T'$. In our setting, $T = 19$ and $T' = 39$.

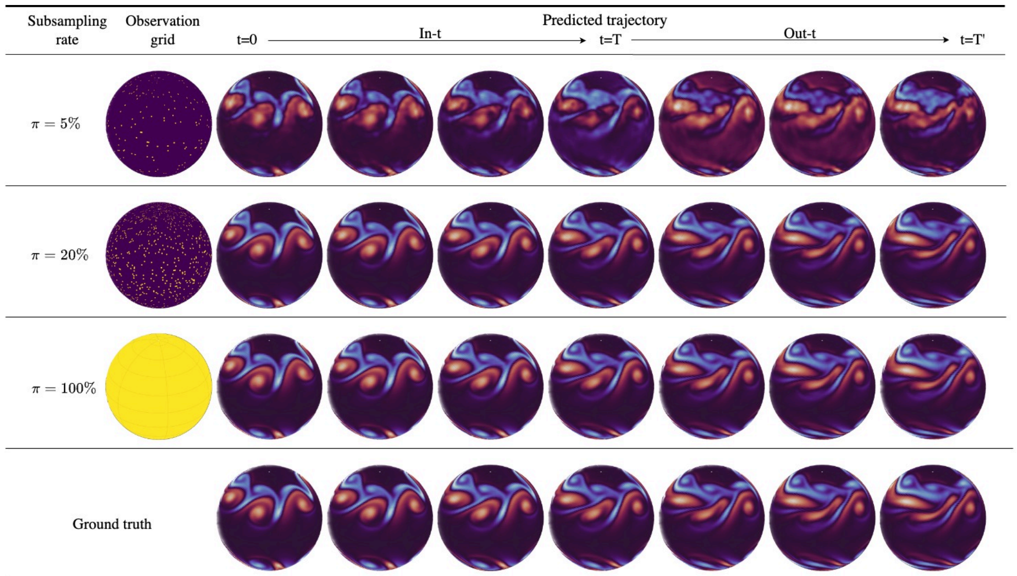


Figure 12: Prediction MSE per frame for CORAL on *Shallow-Water* with its corresponding training grid $\mathcal{X}$. Each row corresponds to a different sampling rate and the last row is the ground truth. The predicted trajectory is predicted from $t = 0$ to $t = T'$. In our setting, $T = 19$ and $T' = 39$.

E.3 Geometric design

We provide in Figure 13, Figure 14, Figure 15 visualization of the predicted values of CORAL on *NACA-Euler*, *Pipe* and *Elasticity*.

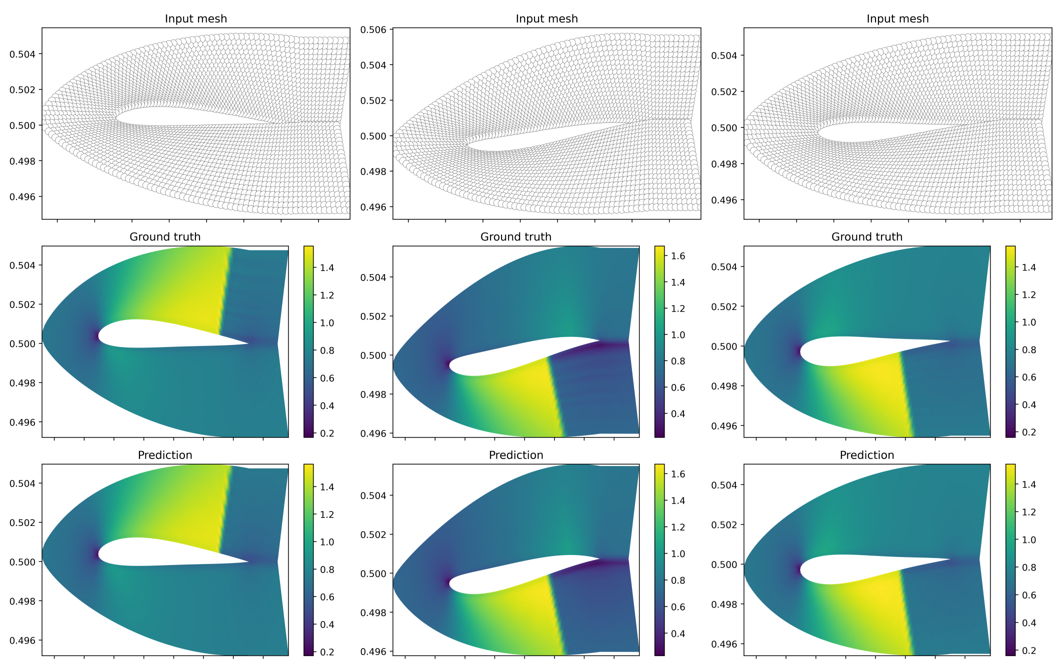


Figure 13: CORAL predictions on *NACA-Euler*

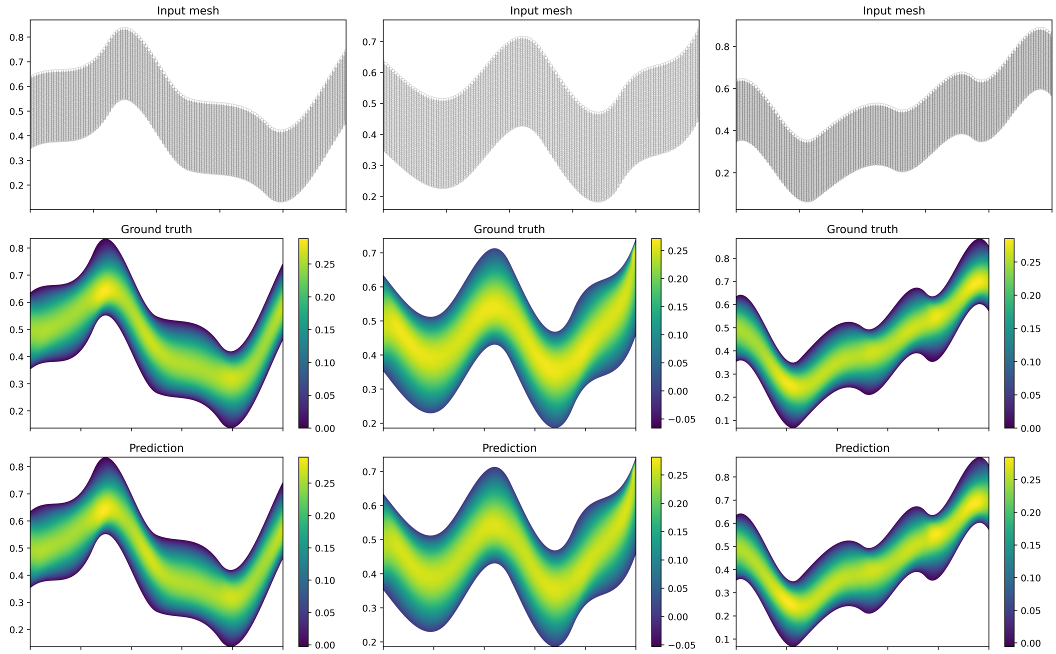


Figure 14: CORAL predictions on *Pipe*

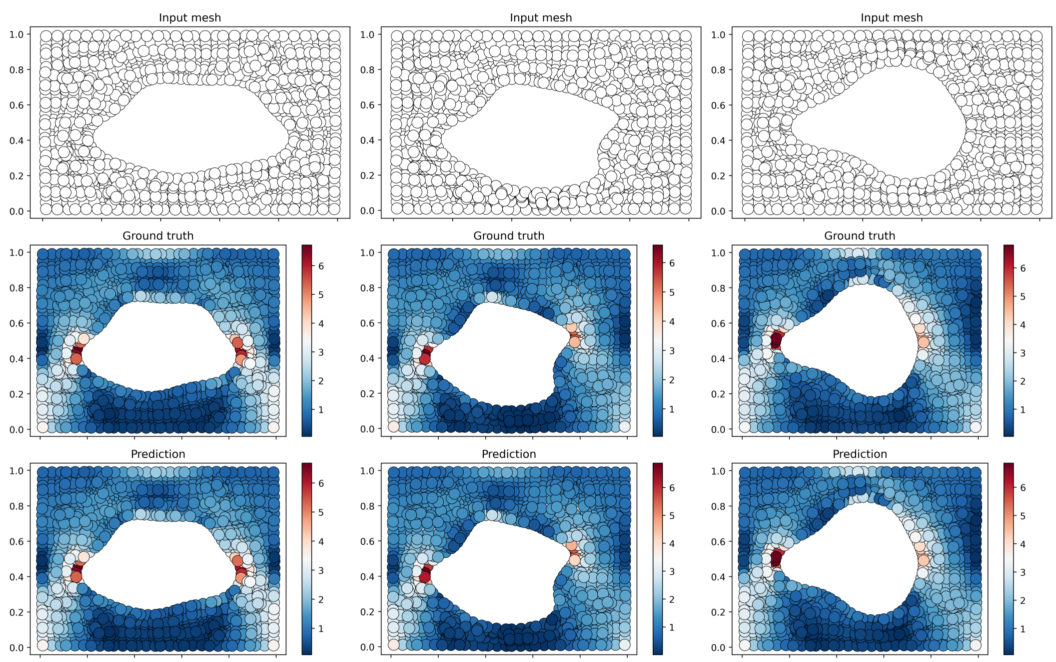


Figure 15: CORAL predictions on *Elasticity*