

A DISCUSSION OF METHODS

FNO can be regarded as a member of the family of Vision Mixers. The reason is that a component in an iteration in Eq. (2) can be written as $(R_\phi \cdot \mathcal{F}(v))(x) = R_\phi \cdot \sum_i e^{-2\pi i \langle x, x_i \rangle} v(x_i)$, because in the equispaced scenarios, x_i can be regarded as lying on the same grids as k after scaling. The kernel κ_ϕ is parameterized by $\kappa_\phi(x, x_i, v(x), v(x_i)) = e^{-2\pi i \langle x, x_i \rangle}$, and the matrix multiplication of R_ϕ also performs mixing on channels. Besides, the inverse Fourier transform can also be regarded as token mixing layers, or so-called token demixing layers (Guibas et al., 2021).

Applicability of Layer-Norm. As shown in Fig. 2, besides the comparison of the proposed interpolation operator with the traditional ones, a notable difference between the original FNO and FNO layers in our Vision Mixer architecture is the applicability of normalization layers (Layer-Norm) which is usually used in Vision Mixers’ architecture. FNO cannot adapt Layer-Norm layers, because the change of resolution will make the trained normalization parameters and spatial points disagree with each other. In comparison, the resampled equispaced points are fixed in our architecture, no matter how the discretization of the input changes. Therefore, the normalization layers can be added, in a similar way to Vision Mixers, bringing considerable improvements (See Sec. B.5).

Mesh invariance. In the intermediate layers, which adopt equispaced FNO, the resampled points are fixed in both training and inference process, invariant to the input meshes. In the interpolation layers, the operator $\mathcal{H}_\eta(a)$ is discretization-invariant because the kernel can be inductively obtained by the newly observed signals $a(x)$, its coordinate x and resampled spatial points’ relative coordinates $x_j - x$. In the same way, $\mathcal{H}'_\zeta(a)$ is also mesh-invariant. This allows the NFS to achieve zero-shot mesh-invariant inference ability, which is demonstrated in Sec. ??.

Complexity analysis. The complexity of FNO is $O(n_s \log n_s + n_s k_{\max})$. In the interpolation layers, because the interpolated values of resampled points are determined by their neighbors, we set the size of each resampled point’s neighborhood in \mathcal{G} and observed non-equispaced points’s neighborhood in \mathcal{G}' as $|\mathcal{N}(x_i)| \approx |\mathcal{N}(x_j)| \leq c \log(n_s)$, for $1 \leq i \leq n_s, 1 \leq j \leq m_s$. And in this way, the sparsity of the interpolation matrix reduces the complexity of the two interpolation layers to $O(c \cdot n_s \log n_s + c \cdot m_s \log n_s)$. If we set the resampled points number as n_s , the overall complexity is $O(2c \cdot n_s \log n_s + n_s \log n_s + n_s k_{\max}) \sim O(n_s \log n_s + n_s k_{\max})$.

Relation to Vision Mixer. The interpolation can be compared to patchwise embedding in Vision Mixers. For example, MLP MIXER learns the token mixing patterns adaptively with a feed-forward network, but the high resolution of input images does not permit the global mixing of tokens due to the complexity of $O(n_s^2)$. Therefore, the input images are firstly rearranged into patches, with each patch containing n_p pixels. In this way, the complexity is reduced to $O(n_s^2/n_p^2)$, enabling feasible token mixing. The patchwise embedding is very similar to interpolating the values on resampled points, as the former one first chooses patches’ centers as n_s^2/n_p^2 resampled points, and ‘interpolates’ the resampled points by lifting the embedding dimension and the rearranging of their neighbors’ values as the interpolated values, rather than using a kernel.

Relation to multipole graph model. The adaptively learned interpolation layer in NFS has a similar formulation of multipole graph models (Li et al., 2020b). In multipole graph models, the high-level nodes aggregate messages from their low-level neighbors as $v^{\text{High}}(x_j) = \frac{1}{|\mathcal{N}(x_j)|} \sum_{x_i \in \mathcal{N}(x_j)} v^{\text{Low}}(x_i) h_\eta(x_j, x_i, a(x_j), a(x_i))$. Compared to multipole graph models, the values of high-level resampled equispaced nodes are approximated with low-level non-equispaced nodes’ values in NFS, but nodes’ values of low levels are given in multipole graphs. This causes differences in multipole graph models’ message-passing and NFS’s interpolation: In the former one, messages flow circularly among different levels of nodes, while in NFS, messages only exchange twice between the nodes of two levels – one is from low-level non-equispaced nodes to high-level resampled equispaced nodes, and the other is the opposite.

B APPROXIMATION THEOREM

Proof of Theorem 3.1. Our proof is mostly based on Chen & Chen (1995) and Kovachki et al. (2021). For notation simplicity, in the proof, we directly write $\mathcal{H}_\eta(a)$ as \mathcal{H}_η as the linear operator.

Lemma B.1. *Let \mathcal{X} be a Banach space, and $\mathcal{U} \subseteq \mathcal{X}$ a compact set, and $\mathcal{K} \subset \mathcal{X}$ a dense set. Then, for any $\epsilon > 0$, there exists a number $n \in \mathbb{N}$, and a series of continuous, linear functionals $G_1, G_2, \dots, G_n \in C(\mathcal{U}; \mathbb{R})$, and elements $\varphi_1, \dots, \varphi_n \in \mathcal{K}$, such that*

$$\sup_{u \in \mathcal{U}} \|v - \sum_{j=1}^n G_j(v) \varphi_j\|_{\mathcal{X}} \leq \epsilon \quad (7)$$

The proof is given in **Lemma 7.** in Kovachki et al. (2021), and **Theorem 3.** and **4.** for reference .

Theorem B.2. *Let $D \subseteq \mathbb{R}^d$ be compact domain. Let \mathcal{U} be a separable Banach space of real-valued functions on D , such that $C(D, \mathbb{R}) \subseteq \mathcal{U}$ is dense. Suppose $\mathcal{U} = L^p(D; \mathbb{R})$ for any $1 < p < \infty$. ν is a probability measure supported on \mathcal{U} and assume that, $\mathbb{E}_{v \sim \nu} \|v\|_{\mathcal{U}} < \infty$ for any $v \in \mathcal{U}$. μ is a probabilistic measure supported on D , which defines the inner product of Hilbert space \mathcal{U} as $\langle f, g \rangle_{\mathcal{U}} = \int_D f(\mathbf{x})g(\mathbf{x})d\mu(\mathbf{x})$. Then, there exists a neural network $h_\eta : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ whose activation functions are of the Tauber-Wiener class, such that*

$$\|v - \mathcal{H}(v)\|_{\mathcal{U}} \leq \epsilon,$$

where $\mathcal{H}(v)(\mathbf{x}) = \int_D h_\eta(\mathbf{x}, \mathbf{y})v(\mathbf{y})d\mu(\mathbf{y})$.

Proof. Since \mathcal{U} is a Polish space, we can find a compact set \mathcal{K} , such that $\nu(\mathcal{U} \setminus \mathcal{K}) \leq \epsilon$. Therefore, **Lemma B.1** can be applied, to find a number $n \in \mathbb{N}$, a series of continuous linear functionals $G_j \in C(\mathcal{U}; \mathbb{R})$ and functions $\varphi_j \in C(D; \mathbb{R})$ such that

$$\sup_{v \in \mathcal{K}} \|v - \sum_{j=1}^n G_j(v) \varphi_j\|_{\mathcal{U}} \leq \epsilon.$$

Denote $\hat{\mathcal{H}}_n(v) = \sum_{j=1}^n G_j(v) \varphi_j$, and let $1 < q < \infty$ be the Hölder conjugate of p . Since $\mathcal{U} = L^p(D; \mathbb{R})$, by Riesz Representation Theorem, there exists functions $g_j \in L^q(D; \mathbb{R})$, such that $G_j(v) = \int_D v(\mathbf{x})g_j(\mathbf{x})d\mu(\mathbf{x})$ for $j = 1, \dots, n$ and $v \in L^p(D; \mathbb{R})$. By density of $C(D; \mathbb{R})$ in $L^q(D; \mathbb{R})$, we can find functions $\psi_1, \dots, \psi_n \in C(D; \mathbb{R})$, such that

$$\sup_{j \in \{1, \dots, n\}} \|\psi_j - g_j\|_{L^q(D; \mathbb{R})} \leq \epsilon/n.$$

Then, we define $\tilde{\mathcal{H}}_n : L^p(D; \mathbb{R}) \rightarrow C(D; \mathbb{R})$ by

$$\tilde{\mathcal{H}}_n(v) = \sum_{j=1}^n \int_D \psi_j(\mathbf{y})v(\mathbf{y})d\mu(\mathbf{y})\varphi_j(\mathbf{x}).$$

For the universal approximation (density) (Hornik et al., 1989) of neural networks, we can find a Multi-layer Feedforward network $h_\eta : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ whose activation functions are of the Tauber-Wiener class, such that

$$\sup_{\mathbf{x}, \mathbf{y} \in D} |h_\eta(\mathbf{x}, \mathbf{y}) - \sum_{j=1}^n \psi_j(\mathbf{y})\varphi_j(\mathbf{x})| \leq \epsilon.$$

Let $\mathcal{H}_\eta(\mathbf{x}) = \int_D h_\eta(\mathbf{x}, \mathbf{y})v(\mathbf{y})d\mu(\mathbf{y})$. Then, there exists a constant $C_1 > 0$, such that

$$\|\hat{\mathcal{H}}_n(v) - \mathcal{H}(v)\|_{L^p(D; \mathbb{R})} \leq C_1(\|\hat{\mathcal{H}}_n(v) - \tilde{\mathcal{H}}_n(v)\|_{L^p(D; \mathbb{R})} + \|\tilde{\mathcal{H}}_n(v) - \mathcal{H}(v)\|_{L^p(D; \mathbb{R})}).$$

For the first term, there is a constant $C_2 > 0$, such that

$$\begin{aligned} \|\hat{\mathcal{H}}_n(v) - \tilde{\mathcal{H}}_n(v)\|_{L^p(D; \mathbb{R})} &\leq C_2 \sum_{j=1}^n \left\| \int_D v(\mathbf{y})(g_j(\mathbf{y}) - \psi_j(\mathbf{y}))d\mu(\mathbf{y})\varphi_j \right\|_{L^p(D; \mathbb{R})} \\ &\leq C_2 \sum_{j=1}^n \|v(\mathbf{y})\|_{L^p(D; \mathbb{R})} \|g_j(\mathbf{y}) - \psi_j(\mathbf{y})\|_{L^q(D; \mathbb{R})} \|\varphi_j\|_{L^p(D; \mathbb{R})} \\ &\leq C_3 \epsilon \|v(\mathbf{y})\|_{L^p(D; \mathbb{R})}, \end{aligned}$$

for some $C_3 > 0$. And for the second term,

$$\begin{aligned} \|\tilde{\mathcal{H}}_n(v) - \mathcal{H}(v)\|_{L^p(D;\mathbb{R})} &= \left\| \int_D v(\mathbf{y}) \left(\sum_{j=1}^n \psi_j(\mathbf{y}) \varphi_j(\cdot) - h_\eta(\cdot, \mathbf{y}) \right) d\mu(\mathbf{y}) \right\|_{L^p(D;\mathbb{R})} \\ &\leq |D| \epsilon \|v\|_{L^p(D;\mathbb{R})}, \end{aligned}$$

Therefore, there is a constant $C > 0$, such that

$$\int_{\mathcal{U}} \|\hat{\mathcal{H}}_n(v) - \tilde{\mathcal{H}}_n(v)\|_{\mathcal{U}} d\nu(v) \leq \epsilon C \mathbb{E}_{v \sim \nu} \|v\|_{\mathcal{U}}$$

. Because of the assumption that $\mathbb{E}_{v \sim \nu} \|v\|_{\mathcal{U}} < \infty$, and ϵ is arbitrary, then

$$\|v - \mathcal{H}(v)\|_{\mathcal{U}} \leq \|v - \hat{\mathcal{H}}_n(v)\|_{\mathcal{U}} + \|\hat{\mathcal{H}}_n(v) - \mathcal{H}(v)\|_{\mathcal{U}},$$

the proof is complete. \square

Corollary B.3. Define $\mathcal{H}_\eta(v) = \int_D h_\eta(\mathbf{x} - \mathbf{y}, \mathbf{x}, a(\mathbf{y})) v(\mathbf{y}) d\mu(\mathbf{y})$, the interpolation operator can also approximate v to any precision ϵ .

Proof. We use a one-layer neural network $h_\eta : D \times D \rightarrow \mathbb{R}$ as an example, which is defined as $h_\eta(\mathbf{x}, \mathbf{y}, a(\mathbf{y})) = \sigma(\sum_{i=1}^d w_{x,i} x^{(i)} + w_{y,i} y^{(i)} + b)$. We can rewrite it as

$$h_\eta = \sigma\left(\sum_{i=1}^d w_{x,i} (x^{(i)} - y^{(i)}) + (w_{y,i} + w_{x,i}) y^{(i)} + \sum_{j=1}^{d_a} w_{a,j} a^{(j)}(\mathbf{y}) + b\right),$$

where $w_{a,j} = 0$. \square

Corollary B.4. The *Theorem B.2* and *Corollary B.3* can be extended for $v : D \rightarrow \mathbb{R}^{d_v}$, where $d_v > 1$.

Proof. As $v = (v^{(1)}, v^{(2)}, \dots, v^{(d_v)})$, for each $v^{(j)}$, a single neural network can be used for approximation. Moreover, in implementation, we make h_η fully-connected, to improve the expressivity. \square

Remark. As $\sum_{\mathbf{x}_i \in \mathbf{X}} v(\mathbf{x}_i) h_\eta(\mathbf{x} - \mathbf{x}_i, \mathbf{x}_i, a(\mathbf{x}_i))$ is the unbiased estimation of $\mathbb{E}_{\mathbf{y} \sim \mu} (h_\eta(\mathbf{x}, \mathbf{y}) v(\mathbf{y}))$, we use the Equation. (6) for the approximation.

B.1 BENCHMARK METHOD DESCRIPTION

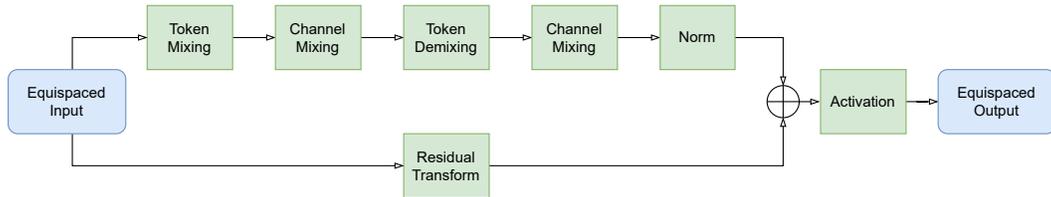


Figure B1: The architecture of Vision Mixers.

Vision Mixers. We provide a framework for vision mixers as PDE solvers, including VIT, MLP-MIXER, FNET, GFN, FNO, PFNO and our NFS. The intermediate architecture of mixing layers is shown in Fig. B1. The code of our framework will be released soon. And the resampling and back-sampling methods are stacked before ‘Equispaced Input’ and ‘Equispaced Output’. In this way, the description of the Vision Mixers included in our framework can be described by different modules, as shown in Table. B1. All the trials on Vision Mixers set *embedding size* as 32, *batch size* as 4, *layer number* of the intermediate equispaced mixing layers as 2. In FNO and PFNO, the truncated K_{\max} is set as 16. The *patch size* of Vision Mixers with patchwise embedding are set as [4, 2] in 1-d PDEs and [4, 4, 2] in 2-d PDEs. The interpolation layers in NFS are composed of one layer of feed-forward network whose perceptron unit is equal to $4 \times \text{embedding size}$ of the model.

Table B1: Description of Vison Mixers in the unifying framework module by module.

Modules	VIT	MLPMIXER	FNET	GFN	FNO	PFNO	NFS
Resampling	Patchwise Embedding	Patchwise Embedding	Patchwise Embedding	Patchwise Embedding	Identity	Patchwise Embedding	Interpolation
Token Mixing	Attention	MLP	Fourier	Fourier	Fourier	Fourier	Fourier
Channel Mixing	Linear	Linear	Linear	Elementwise Product	Low Frequency MatMultiply	Low Frequency MatMultiply	Low Frequency MatMultiply
Token Demixing	Identity	Identity	Identity	Inverse Fourier	Inverse Fourier	Inverse Fourier	Inverse Fourier
Channel Mixing	Identity	Identity	Identity	Linear	Identity	Identity	Identity
Normalization	LayerNorm	LayerNorm	Complex LayerNorm	LayerNorm	Identity	LayerNorm	LayerNorm
Residual	Identity	Identity	Identity	Identity	1x1 Conv	1x1 Conv	1x1 Conv
Activation	Gelu	Gelu	Complex Gelu	Gelu	Gelu	Gelu	Gelu
Back Sampling	Linear+ Rearrange	Linear+ Rearrange	Linear+ Rearrange	Linear+ Rearrange	Identity	Linear+ Rearrange	Interpolation

DeepONet Variants. Since vanilla DeepONet uses MLP as Branch Net, it cannot be implemented in such a high-resolution dataset, because for a resolution like the trial ($n_s = 4096, n_t = 10, n'_t = 10$), DeepONet assigns each data point a weight parameter in a single MLP, leading the MLP’s parameter number reaches $O(n_s^2 n_t^2) \approx 40960^2$ in a single Branch Net, which is infeasible in practice. In the original paper, the spatial point’s number in the experiments is set as 40, far less than in the recent Neural Operator’s evaluation protocol.

One feasible alternative is to use other architecture to replace the original MLP, thus allowing DeepONet to handle high-resolution data. For example, CNN and Vit. Therefore, we here conduct further experiments on the three equations in the context, to evaluate DeepONet-U (using UNet as the Branch Net) and DeepONet-V (using Vit as the Branch Net) as two variants of vanilla DeepONet for comparison. Note that the architecture of variants of DeepONet are all limited to equispaced data.

Graph Spatio-Temporal Models. The evaluated graph spatio-temporal neural networks are based on recurrent neural networks for dynamics modeling, where the spatial dependency is modeled by graph neural networks. The spatial and temporal modules for AGCRN, DCRNN and GCGRU are shown in Table. B2. MPPDE used different architecture, with the pushforward trick used for taining, with *rolling* equaling 1 and *time window* equaling to 10 . All the trials on these graph spatio-temporal models set *embedding size* as 64, except MPPDE as 128. *Batch size* is set as 4. When the graph convolution needs multi-hop message-passing, we set the hop as 2. For MPPDE, the layer number of GNNs is 6. The *embedding dimension* in AGCRN is set as 2.

Table B2: Description of different graph spatio-temporal models

Methods	Spatial module	Temporal module
GCGRU Seo et al. (2016)	Cheb Conv Defferrard et al. (2017)	GRU
DCRNN Li et al. (2018)	Diff Conv Atwood & Towsley (2016)	GRU
AGCRN Bai et al. (2020)	Node Similarity Bai et al. (2020)	GRU

B.2 DATA GENERATION

Burgers’ Equation. The initial condition $u_0(x)$ is generated according to $u_0 \sim N(0.625(-\Delta + 25I)^{-2})$ with periodic boundary conditions. ν is set as 0.01. $x \in [0, 1]$ and $t \in [0, 1]$. The spatial resolution is 1024, and time resolution is 200. The dataset generation follows FNO’s protocol, which can be downloaded from its source code on official Github.

KdV Equation. The equation is written as

$$\partial_t u(x, t) + 3\partial_x u^2(x, t) + \partial_x^3 u(x, t) = 0, \quad (8)$$

where $x \in [0, 1]$. The initial condition $u_0(x)$ is calculated as

$$u(x, 0) = \sum_{i=1}^K 0.5c_i \cos(0.5\sqrt{c_i + b_i}x - a_i)$$

where $c_i \sim N(0, \sigma_i)$, and $a_i, b_i > 0$. The spatial resolution is 1024. The dataset is generated by `scipy` package, with `fftpack.diff` used as pseudo-differential method and `odeint` used as forward Euler method.

Darcy Flow. The equation is written as

$$\begin{aligned} -\nabla(a(\mathbf{x})\nabla u(\mathbf{x})) &= f(\mathbf{x}) & \mathbf{x} &\in (0, 1)^2 \\ u(\mathbf{x}) &= 0 & \mathbf{x} &\in \partial[0, 1]^2 \end{aligned} \quad (9)$$

The original resolution is 256×256 . $a(\mathbf{x})$ is generated by Gaussian random field, and we directly establish the operator to learn the mapping of a to u .

NS Equation. Our generation of NS Equation is based on FNO’s Appendix. A.3.3, which is

$$\begin{aligned} \partial_t w(\mathbf{x}, t) + u(\mathbf{x}, t) \cdot \nabla w(\mathbf{x}, t) &= \nu \Delta w(\mathbf{x}, t) + f(\mathbf{x}), \\ \nabla \cdot u(\mathbf{x}, t) &= 0, \quad w(\mathbf{x}, 0) = w_0(\mathbf{x}), \end{aligned} \quad (10)$$

where $\mathbf{x} \in [0, 1]^2, t \in [0, 1]$. u is the velocity field, $w = \nabla \times u$ is the vorticity, w_0 is the initial vorticity, $\nu \in \mathbb{R}^+$ is the viscosity coefficient, and f is the forcing function which is kept fixed. The original spatial resolution is 128×128 , and time resolution is 200.

B.3 COMPLETE RESULTS ON MODEL COMPARISON

Table. B3 and Fig. B2 is a brief demonstration of the model comparison.

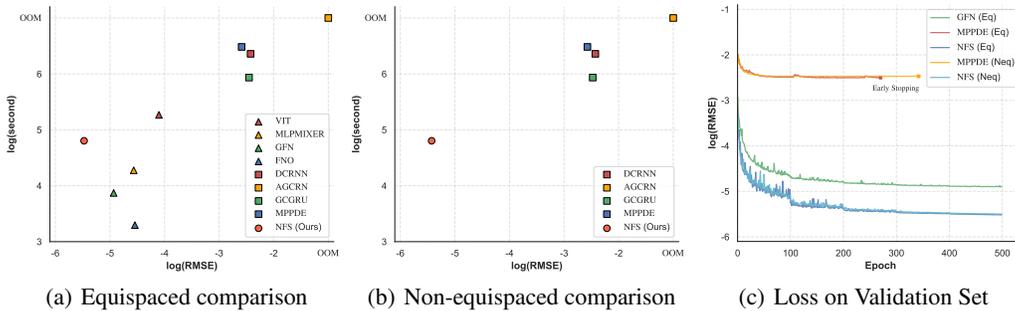


Figure B2: In (a) and (b), ‘ \triangle ’ represents Vision Mixers, ‘ \square ’ represents graph spatio-temporal models and ‘ \circ ’ is the proposed NFS. In (c), ‘Eq’ and ‘Neq’ mean the methods are trained in equispaced and non-equispaced scenarios respectively.

Table B3: MAE($\times 10^{-3}$) comparison with graph spatio-temporal benchmarks.

	Burgers’ ($n_t = 10$)			Darcy Flow			NS ($n_t = 1$)			NS ($n_t = 10$)		
	n_s	n'_t										
	512	512	256	4096	16384	1024	4096	4096	1024	4096	4096	1024
	10	40	20	1	1	1	10	40	20	10	40	20
DCRNN	2.6122	8.5880	4.6126	1.7629	OOM	1.8146	30.6756	88.3382	52.1290	8.7025	59.6602	27.1069
AGCRN	4.6667	15.6143	10.4900	1.7336	OOM	1.6938	OOM	OOM	59.9393	OOM	OOM	42.4197
GCGRU	1.6643	5.7653	3.1400	1.7403	OOM	1.7633	28.8537	85.9303	49.9352	6.3570	57.2493	21.3537
MPPDE	1.1271	4.1213	2.4554	0.5608	0.6384	0.6673	8.9810	54.2387	20.7453	5.4353	42.3057	17.5902
NFS	0.1085	0.1983	0.1634	0.1430	0.2379	0.1727	2.1992	4.7865	3.9178	0.9335	3.2768	1.8239

Complete results on the four Equations. Table. B4 give the performance comparison on Darcy flow of both equispaced and non-equispaced scenarios. Table. B5 and B6 gives performance comparison

in equispaced scenarios on the other three time-dependent problems. Table. B7 and B8 gives performance comparison in non-equispaced scenarios on the other three time-dependent problems. In all the tasks except Darcy Flow, the depth of layer is set as 2, and $k_{\max} = 16$ in both NFS and FNO. However, we find in Darcy Flow, k_{\max} should be set much larger, or the loss will not decrease. In the reported results, $k_{\max} = 32, 64, 128$ in Darcy Flow.

Table B4: Performance comparison on Darcy Flow.

	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)
	Darcy Flow ($r = 64$)		Darcy Flow ($r = 128$)		Darcy Flow ($r = 256$)	
VIT	0.5073 \pm 0.0411	0.8468 \pm 0.0432	0.9865 \pm 0.0002	1.6195 \pm 0.0007	1.1078 \pm 0.0021	1.8444 \pm 0.0023
MLPMIXER	0.4970 \pm 0.0021	0.8228 \pm 0.0034	0.8909 \pm 0.0099	1.4221 \pm 0.0118	0.9125 \pm 0.0024	1.6459 \pm 0.0032
GFN	0.4739 \pm 0.0016	0.8345 \pm 0.0019	0.8659 \pm 0.0046	1.4237 \pm 0.0071	0.9618 \pm 0.0124	1.6139 \pm 0.0128
FNO	0.4289 \pm 0.0051	0.7740 \pm 0.0046	0.7086 \pm 0.0045	0.1324 \pm 0.0019	0.9075 \pm 0.0051	1.4940 \pm 0.0046
NFS	0.1497 \pm 0.0005	0.1962 \pm 0.0007	0.2254 \pm 0.0007	0.7245 \pm 0.0009	0.4216 \pm 0.0033	0.8578 \pm 0.0041
	Darcy Flow ($n_s = 1024$)		Darcy Flow ($n_s = 4096$)		Darcy Flow ($n_s = 16384$)	
DCRNN	1.8146 \pm 0.0060	2.6352 \pm 0.0029	1.7629 \pm 0.0003	2.5760 \pm 0.0001	OOM	OOM
AGCRN	1.6938 \pm 0.0001	2.4440 \pm 0.0001	1.7336 \pm 0.0001	2.4167 \pm 0.0001	OOM	OOM
GCGRU	1.7633 \pm 0.0001	2.5696 \pm 0.0001	1.7403 \pm 0.0001	2.5363 \pm 0.0001	OOM	OOM
MPPDE	0.6673 \pm 0.0009	0.9290 \pm 0.0012	0.5608 \pm 0.0053	0.8424 \pm 0.0051	0.6384 \pm 0.0005	0.8748 \pm 0.0005
NFS	0.1727 \pm 0.0047	0.2311 \pm 0.0066	0.1430 \pm 0.0007	0.1914 \pm 0.0014	0.2379 \pm 0.0007	0.3489 \pm 0.0009

Table B5: Performance comparison with Vision Mixer benchmarks on different equations ($n_t = 1$). Validation loss on Burgers' ($n_t = 1$) of VIT, GFN, and FNO does not converge. The results show that the early-stopping occurs in the begining of training.

	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)
Vision Mixers	Burgers' ($r = 512, n_t' = 10$)		Burgers' ($r = 512, n_t' = 40$)		Burgers' ($r = 1024, n_t' = 20$)	
VIT	201.6539 \pm 0.5284	231.9138 \pm 0.8403	183.6696 \pm 0.3015	210.6237 \pm 0.6767	195.5858 \pm 0.7706	224.4712 \pm 1.2472
MLPMIXER	201.6547 \pm 0.0671	231.9163 \pm 0.0263	183.6535 \pm 0.0599	210.6160 \pm 0.0305	195.5960 \pm 0.0240	224.4791 \pm 0.0132
GFN	201.6557 \pm 0.9513	231.9122 \pm 0.9535	183.6674 \pm 0.4893	210.6165 \pm 0.4831	195.5918 \pm 0.0471	224.4736 \pm 0.0681
FNO	201.6527 \pm 1.1415	231.9119 \pm 1.6747	183.6696 \pm 0.3015	210.6299 \pm 0.4983	195.5902 \pm 0.9304	224.4723 \pm 0.9230
NFS	0.1806 \pm 0.0005	0.2669 \pm 0.0010	0.3570 \pm 0.0008	0.5340 \pm 0.0009	0.4344 \pm 0.0014	0.6092 \pm 0.0017
	KdV ($r = 512, n_t' = 10$)		KdV ($r = 512, n_t' = 40$)		KdV ($r = 1024, n_t' = 20$)	
VIT	0.2808 \pm 0.0006	0.3938 \pm 0.0009	0.3428 \pm 0.0012	0.6832 \pm 0.0016	0.3066 \pm 0.0003	0.5461 \pm 0.0003
MLPMIXER	0.2732 \pm 0.0054	0.4259 \pm 0.0088	0.3336 \pm 0.0045	0.5923 \pm 0.0081	0.2872 \pm 0.0005	0.5235 \pm 0.0006
GFN	0.2587 \pm 0.0032	0.3490 \pm 0.0056	0.3086 \pm 0.0223	0.5952 \pm 0.0338	0.2011 \pm 0.0074	0.3464 \pm 0.0063
FNO	0.2619 \pm 0.0069	0.3849 \pm 0.0107	0.5608 \pm 0.0053	0.8424 \pm 0.0051	0.3925 \pm 0.0079	0.4623 \pm 0.0087
NFS	0.2514 \pm 0.0008	0.3776 \pm 0.00011	0.4522 \pm 0.0013	0.6290 \pm 0.0022	0.2254 \pm 0.0007	0.0745 \pm 0.0010
	NS ($r = 64, n_t' = 10$)		NS ($r = 64, n_t' = 40$)		NS ($r = 128, n_t' = 20$)	
VIT	9.3797 \pm 0.0421	12.9291 \pm 0.0703	22.8565 \pm 0.0935	29.1130 \pm 0.1428	15.7398 \pm 0.0757	20.6927 \pm 0.0664
MLPMIXER	7.5246 \pm 0.0080	10.4762 \pm 0.0096	15.8632 \pm 0.0375	20.1522 \pm 0.0604	14.9360 \pm 0.0305	19.3268 \pm 0.0635
GFN	3.5524 \pm 0.0057	4.7071 \pm 0.0088	10.2250 \pm 0.0331	13.0451 \pm 0.0704	6.3976 \pm 0.00345	8.2685 \pm 0.297
FNO	3.3425 \pm 0.0007	5.2566 \pm 0.0008	8.9857 \pm 0.0010	14.0171 \pm 0.0023	4.4627 \pm 0.0004	6.3047 \pm 0.0004
NFS	1.7425 \pm 0.0017	2.2847 \pm 0.0022	4.7882 \pm 0.0066	6.1508 \pm 0.0042	2.6988 \pm 0.0005	3.5121 \pm 0.0006

NFS fails to model the non-equispaced Burgers' Equation when n_t is set as 1, in which the performance is far from it can achieve in equispaced scenarios. Such problem will be our future work.

Table B6: Performance comparison with Vision Mixer benchmarks on different equations ($n_t = 10$).

Vision Mixers	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)
	Burgers' ($r = 512, n_t' = 10$)		Burgers' ($r = 512, n_t' = 40$)		Burgers' ($r = 1024, n_t' = 20$)	
VIT	0.5042 \pm 0.0114	0.7667 \pm 0.0225	2.4269 \pm 0.0288	3.7728 \pm 0.0431	1.5327 \pm 0.0314	2.4093 \pm 0.0408
MLPMIXER	0.1973 \pm 0.0070	0.2600 \pm 0.0097	0.4210 \pm 0.0084	0.5844 \pm 0.0101	0.3303 \pm 0.0077	0.4473 \pm 0.0086
GFN	0.2383 \pm 0.0082	0.3066 \pm 0.0114	0.4187 \pm 0.0079	0.5407 \pm 0.0090	0.3500 \pm 0.0062	0.4489 \pm 0.0081
FNO	0.0978 \pm 0.0019	0.1287 \pm 0.0023	0.1815 \pm 0.0009	0.2410 \pm 0.0011	0.1430 \pm 0.0009	0.1871 \pm 0.0010
NFS	0.0958 \pm 0.0015	0.1347 \pm 0.0022	0.1708 \pm 0.0006	0.2351 \pm 0.0009	0.1474 \pm 0.0026	0.1957 \pm 0.0034
	KdV ($r = 512, n_t' = 10$)		KdV ($r = 512, n_t' = 40$)		KdV ($r = 1024, n_t' = 20$)	
	VIT	0.2066 \pm 0.0027	0.3525 \pm 0.0049	0.2376 \pm 0.0022	0.5521 \pm 0.0036	0.1897 \pm 0.0003
MLPMIXER	0.2152 \pm 0.0023	0.3686 \pm 0.0039	0.2497 \pm 0.0017	0.5400 \pm 0.0029	0.2062 \pm 0.0007	0.4429 \pm 0.0012
GFN	0.1530 \pm 0.0004	0.2607 \pm 0.0006	0.2691 \pm 0.0007	0.5451 \pm 0.0014	0.1984 \pm 0.0002	0.3869 \pm 0.0003
FNO	0.3230 \pm 0.0035	1.1105 \pm 0.0061	0.9605 \pm 0.0024	2.7500 \pm 0.0055	0.5929 \pm 0.0020	1.6473 \pm 0.0033
NFS	0.0678 \pm 0.0002	0.1214 \pm 0.0003	0.2709 \pm 0.0009	0.5122 \pm 0.0013	0.1576 \pm 0.0003	0.3114 \pm 0.0005
	NS ($r = 64, n_t' = 10$)		NS ($r = 64, n_t' = 40$)		NS ($r = 128, n_t' = 20$)	
	VIT	3.9609 \pm 0.0101	6.0575 \pm 0.0250	12.3433 \pm 0.0342	16.5238 \pm 0.0415	9.3010 \pm 0.0234
MLPMIXER	3.1530 \pm 0.0049	4.4339 \pm 0.0067	7.9291 \pm 0.0038	10.4149 \pm 0.0066	7.7410 \pm 0.0037	10.1934 \pm 0.0082
GFN	1.7396 \pm 0.0016	2.3551 \pm 0.0028	5.4464 \pm 0.0023	7.2130 \pm 0.0032	3.1261 \pm 0.0026	4.1691 \pm 0.0047
FNO	2.4076 \pm 0.0017	3.2861 \pm 0.0024	7.6979 \pm 0.0035	10.6401 \pm 0.0056	3.7001 \pm 0.0034	5.0047 \pm 0.0072
NFS	0.8636 \pm 0.0008	1.2264 \pm 0.0011	3.1122 \pm 0.0020	4.1950 \pm 0.0037	1.8406 \pm 0.0003	2.5620 \pm 0.0005

Table B7: Performance comparison with graph spatio-temporal benchmarks ($n_t = 1$).

Graph Spatio-Temporal Models	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)
	Burgers' ($n_s = 512, n_t' = 10$)		Burgers' ($n_s = 256, n_t' = 20$)		Burgers' ($n_s = 512, n_t' = 40$)	
DCRNN	277.8393 \pm 0.0082	346.1716 \pm 0.0088	292.1712 \pm 0.0280	368.1883 \pm 0.0204	298.4096 \pm 0.0137	373.0938 \pm 0.0186
AGCRN	289.9780 \pm 0.0001	360.9834 \pm 0.0001	272.6697 \pm 0.3404	340.1351 \pm 0.5435	305.4976 \pm 0.2120	376.0804 \pm 0.2385
GCGRU	288.4507 \pm 0.0246	361.1175 \pm 0.0512	294.9075 \pm 0.0005	367.4703 \pm 0.0004	291.0365 \pm 0.0265	365.1668 \pm 0.0827
MPPDE	24.4997 \pm 0.0014	34.5123 \pm 0.0017	25.4357 \pm 0.0002	31.7015 \pm 0.0002	25.3311 \pm 0.0004	33.7808 \pm 0.0005
NFS	16.1860 \pm 0.0016	28.1504 \pm 0.0021	21.1634 \pm 0.0018	33.8976 \pm 0.0018	26.0818 \pm 0.0001	44.7962 \pm 0.0003
	KdV ($n_s = 512, n_t' = 10$)		KdV ($n_s = 256, n_t' = 20$)		KdV ($r = 512, n_t' = 40$)	
	DCRNN	1.6855 \pm 0.0001	3.0875 \pm 0.0001	3.1267 \pm 0.0001	4.8662 \pm 0.0001	5.7387 \pm 0.0001
AGCRN	4.0753 \pm 0.0001	6.8943 \pm 0.0001	5.4107 \pm 0.0001	9.2333 \pm 0.0001	8.4438 \pm 0.0001	13.8677 \pm 0.0001
GCGRU	1.6554 \pm 0.0001	2.6839 \pm 0.0001	3.0677 \pm 0.0001	4.6557 \pm 0.0001	5.8745 \pm 0.0001	9.4528 \pm 0.0001
MPPDE	1.5452 \pm 0.0001	2.6774 \pm 0.0001	2.9929 \pm 0.0007	5.4582 \pm 0.0010	3.0101 \pm 0.0001	4.9946 \pm 0.0001
NFS	0.0816 \pm 0.0012	0.1512 \pm 0.0022	0.1576 \pm 0.0007	0.3114 \pm 0.0018	0.3210 \pm 0.0021	0.6873 \pm 0.0049
	NS ($n_s = 4096, n_t' = 10$)		NS ($n_s = 1024, n_t' = 20$)		NS ($n_s = 4096, n_t' = 40$)	
	DCRNN	30.6756 \pm 0.0001	41.7815 \pm 0.0001	52.1290 \pm 0.0138	69.7019 \pm 0.0032	88.3382 \pm 0.0864
AGCRN	OOM	OOM	59.9393 \pm 0.0001	79.0434 \pm 0.0001	OOM	OOM
GCGRU	28.8537 \pm 0.0019	40.1215 \pm 0.0008	49.9352 \pm 0.0028	67.5623 \pm 0.0014	85.9303 \pm 0.0731	117.9925 \pm 0.0172
MPPDE	8.9810 \pm 0.0014	12.1595 \pm 0.0022	20.7453 \pm 0.0008	32.1098 \pm 0.0018	54.2387 \pm 0.0006	90.0190 \pm 0.0007
NFS	2.1992 \pm 0.0021	2.8280 \pm 0.0033	3.9178 \pm 0.0054	5.0182 \pm 0.0080	4.7865 \pm 0.0042	6.1384 \pm 0.0069

Table B8: Performance comparison with graph spatio-temporal benchmarks ($n_t = 10$).

Graph Spatio-Temporal Models	$\text{MAE}(\times 10^{-3})$ $\text{RMSE}(\times 10^{-3})$		$\text{MAE}(\times 10^{-3})$ $\text{RMSE}(\times 10^{-3})$		$\text{MAE}(\times 10^{-3})$ $\text{RMSE}(\times 10^{-3})$	
	Burgers' ($n_s = 512, n'_t = 10$)		Burgers' ($n_s = 256, n'_t = 20$)		Burgers' ($n_s = 512, n'_t = 40$)	
DCRNN	2.6122±0.0014	3.8435±0.0019	4.6126±0.0015	6.8853±0.0033	8.5880±0.0020	12.7394±0.0037
AGCRN	4.6667±0.0001	6.2791±0.0001	10.4900±0.0009	13.9810±0.0022	15.6143±0.0002	21.0937±0.0001
GCGRU	1.6643±0.0002	2.5074±0.0003	3.1400±0.0010	4.8008±0.0019	5.7653±0.0021	8.9335±0.0028
MPPDE	1.1271±0.0004	1.8838±0.0007	2.4554±0.0003	4.4315±0.0006	4.1213±0.0006	6.1980±0.0009
NFS	0.1085±0.0016	0.1504±0.0021	0.1634±0.0018	0.2328±0.0018	0.1983±0.0001	0.2775±0.0003
	KdV ($n_s = 512, n'_t = 10$)		KdV ($n_s = 256, n'_t = 20$)		KdV ($r = 512, n'_t = 40$)	
DCRNN	2.3196±0.0001	4.1634±0.0001	3.4503±0.0005	5.7450±0.0003	4.9286±0.0010	8.3912±0.0008
AGCRN	3.9350±0.0001	6.1166±0.0001	5.6631±0.0001	8.1191±0.0001	8.2893±0.0002	11.5684±0.0003
GCGRU	1.6643±0.0001	2.5074±0.0001	3.4205±0.0001	5.6873±0.0001	2.5032±0.0002	5.4515±0.0003
MPPDE	1.4967±0.0003	2.6309±0.0002	2.9708±0.0027	5.3811±0.0050	2.4293±0.0006	4.9310±0.0005
NFS	0.0816±0.0012	0.1512±0.0022	0.1576±0.0007	0.3114±0.0018	0.3210±0.0021	0.6873±0.0049
	NS ($n_s = 4096, n'_t = 10$)		NS ($n_s = 1024, n'_t = 20$)		NS ($n_s = 4096, n'_t = 40$)	
DCRNN	8.7025±0.0003	12.5238±0.0002	27.1069±0.0024	39.1259±0.0031	59.6602±0.0177	88.2946±0.0146
AGCRN	OOM	OOM	42.4197±0.0006	60.5375±0.0008	OOM	OOM
GCGRU	6.3570±0.0001	9.7306±0.0002	21.3537±0.0026	32.9674±0.0033	57.2493±0.0085	84.1847±0.0106
MPPDE	5.4353±0.0041	7.8838±0.0037	17.5902±0.0013	25.9372±0.0016	42.3057±0.0066	76.3374±0.0069
NFS	0.9335±0.0011	1.3254±0.0012	1.8239±0.0012	2.5291±0.0008	3.2768±0.0026	4.3988±0.0009

B.4 MESH-INVARIANT EVALUATION AND ARCHITECTURE ANALYSIS

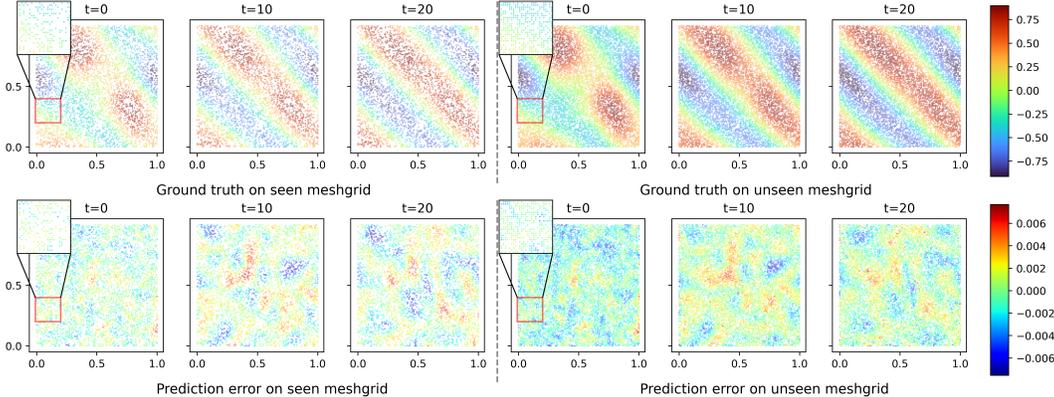


Figure B3: Visualization on non-equispaced NS equation: The training mesh ($n_s = 4096$) differs from the mesh in inference process ($n'_s = 8192$). Appendix. B.9 gives more visualization.

Table B9: Results on NS equation: MAE($\times 10^{-3}$) of NFS and different variants of NFS on seen and unseen meshes. ‘Flex + LN’ is the proposed NFS. ‘Flex’ represents the flexible interpolation layer defined in Eq. (6), ‘LN’ is the Layer-Norm and ‘Gaus’ is the predefined Gaussian interpolation. Appendix. B.4 gives details and results on other equations.

n'_s	$(n_s = 4096, n_t = 10, n'_t = 10)$			$(n_s = 1024, n_t = 10, n'_t = 20)$			$(n_s = 4096, n_t = 10, n'_t = 40)$		
	Flex + LN	Gaus + LN	Flex + LN	Flex + LN	Gaus + LN	Flex + LN	Flex + LN	Gaus + LN	Flex + LN
n_s	0.9335	1.6341	1.2138	1.8239	2.1976	2.5119	3.2768	3.6422	5.6761
$2n_s$	0.9731	2.8589	1.4882	2.3530	3.7465	7.0203	3.5439	3.9092	5.8975
$3n_s$	1.1071	3.4513	1.6384	2.5179	5.7712	7.9177	3.6584	4.2102	6.6622
$4n_s$	1.1015	3.4357	1.6975	2.5919	5.5990	7.1962	3.6608	4.2628	6.6951

Mesh-invariance evaluation. We use $(u(\mathbf{X}, T), u(\mathbf{X}, T'))$ as the training set, and evaluate the model’s performance of mesh-invariant inference ability on \mathbf{X}' , where $|\mathbf{X}'| = n'_s$. \mathbf{X}' is a different mesh with $\mathbf{X} \subseteq \mathbf{X}'$. The visualization results of NS ($n_s = 4096, n'_t = 40$) are shown in Fig. B3. For a fixed n'_s , we randomly sampled different \mathbf{X}' for 100 times, to get the mean errors and standard deviations (given in Appendix. B.4) of different spatial meshes. We can conclude from Table. B9 that (1) The errors on unseen meshes are larger than the errors on seen meshes, showing the overfitting effects. However, the errors on unseen meshes are acceptable, since they are even lower than other models’ prediction error on seen meshes. (2) Larger n'_s leads to higher prediction error because a large number of unseen points are likely to disturb the learned token mixing patterns. On the other hand, NS ($n_s = 1024, n'_t = 10$) implies that small spatial point numbers of training meshes (n_s) hinder model’s generalizing ability on unseen meshes, due to excessive loss of spatial information.

B.5 ARCHITECTURE ANALYSIS

Two modules in NFS differ from FNO. The first is the interpolation layers at the beginning and the end of the architecture. The second is the extra Layer-Norm in the FNO layers, which can be applicable in NFS thanks to its fixed resampled equispaced points, but inapplicable to FNO for preserving its resolution-invariance. We aim to figure out what makes NFS outperform FNO.

Effects of neighborhood sizes. It is widely believed that modeling the long-range dependency among tokens brings improvements (Naseer et al., 2021; Tuli et al., 2021; Mao et al., 2021). By contrast, some local kernel methods demonstrate their superiority (Yang et al., 2019; Liu et al., 2021; Chu

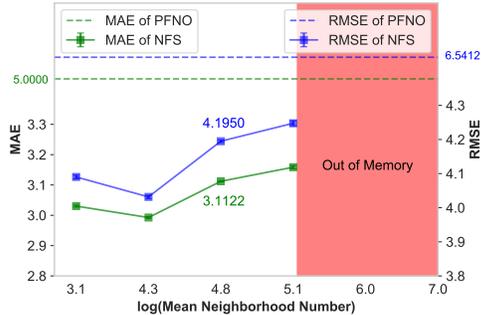


Figure B4: Effects of neighborhood sizes on NS ($r = 64, n'_t = 10, n'_t = 40$).

et al., 2021; Park & Kim, 2022). For this reason, we first conjecture that the large neighborhood sizes in the interpolation layer are conducive to predictive performance. Besides, as demonstrated in Sec. A, the patchwise embedding in Vision Mixers can be an analogy to the resampling and interpolating, so we further establish a patchwise FNO (PFNO), with patch size equaling to 4 and $[4, 4]$ in 1-d and 2-d PDE problems, equivalent to each resampled points aggregating 4 and 16 points in spatial domains in 1-d and 2-d situations respectively. Layer-Norm is stacked in the FNO layers in PFNO, for a fair comparison. Results of Fig. B4 show that the long-range dependency may even compromise the performance, as larger mean neighborhood sizes often cause higher errors. However, no matter how large is the neighbor size, the NFS outperforms PFNO. Therefore, we rule out the possibility of performance gains brought from large neighborhood sizes and suppose that proposed kernel interpolation layers are the key, and is superior to the simple patchwise embedding methods.

The effects of mean neighborhood size on the predictive performance on Burgers' ($n_s = 512, n_t = 10, n'_t = 40$) and KdV ($n_s = 512, n_t = 10, n'_t = 40$) are shown in Fig. B5.

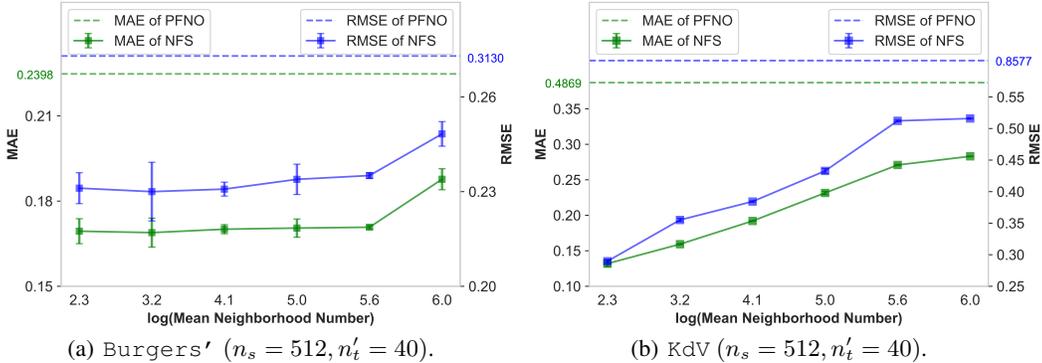


Figure B5: The change of MAE and RMSE of NFS with the increase of neighborhood size on Burgers' ($n_s = 512, n_t = 10, n'_t = 40$) and KdV ($n_s = 512, n_t = 10, n'_t = 40$). PFNO is the baseline.

Benefits from learned interpolation kernel. Since the kernel interpolation is likely to hold the key to improvements, we investigate the performance gains brought from adaptively learned interpolation kernels over the predefined one (See Fig. 2). We use an inflexible Gaussian kernel $h(\mathbf{x}_j - \mathbf{x}_i) = \beta \exp(-(\mathbf{x}_j - \mathbf{x}_i - \boldsymbol{\mu})^T (\boldsymbol{\Gamma})^{-1} (\mathbf{x}_j - \mathbf{x}_i - \boldsymbol{\mu}))$ as a predefined one as discussed in Sec. 3.1, where $\boldsymbol{\Gamma} = \text{diag}(\gamma^{(1)}, \dots, \gamma^{(d)})$, and $\boldsymbol{\mu} \in \mathbb{R}^d, \gamma^{(1)}, \dots, \gamma^{(d)}, \beta \in \mathbb{R}^+$ are learnable parameters. By setting all the other modules and the interpolation neighborhood sizes as the same, we compare performance on different meshes of the two interpolation kernels in Table. B9 (Gaus + LN), where the adaptively learned kernels achieve better accuracy.

Benefits from normalization layers. Previous works demonstrated the normalization is necessary for network architecture, for fast convergence and stable training (Dong et al., 2021; Ba et al., 2016). A notable difference between NFS and FNO is that the Layer-Norm can be implemented in NFS's layers without disabling its discretization-invariance. The improvements brought from the normalization layers are given in Table. B9 (Flex + LN), where the performance gap is obvious on unseen meshes.

B.6 NON-EQUISPACED VISION MIXERS

Since NFS can be regarded as a combination of our interpolation layers with the revised FNO, our interpolation layers can also be implemented in the other Vision Mixers, so that these methods are equipped with the ability to handle non-equispaced data. Details are given in Appendix. B.7. We find that (1) From Table. B12 and Table. 1, the degeneration of performance is obvious in other Vision Mixers. In comparison, FNO as intermediate equispaced layers, truncates the high frequency in its channel mixing and retains the low frequency shared by both resampled and original signals, so the loss of accuracy in non-equispaced scenarios is tiny in our NFS; (2) Although the performance on

unseen meshes is more stable in these non-equispaced Vision Mixers, the performance gap is still large, according to Table. B12 and Table. B9.

Table B10: Mesh-invariant performance of NFS on Burgers’ and KdV equations ($n_t = 10$).

	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)
	Burgers’		KdV	
	$(n_s = 512, n_t = 10, n'_t = 40)$		$(n_s = 512, n_t = 10, n'_t = 40)$	
\mathbf{X}	0.1983 \pm 0.0001	0.2775 \pm 0.0002	0.3210 \pm 0.0021	0.6873 \pm 0.0049
$n'_s = 1.3n_s$	0.2371 \pm 0.0034	0.3143 \pm 0.0041	0.3769 \pm 0.0030	0.7805 \pm 0.0077
$n'_s = 1.7n_s$	0.2898 \pm 0.0113	0.3742 \pm 0.0102	0.4084 \pm 0.0072	0.8419 \pm 0.0174
$n'_s = 2.0n_s$	0.3052 \pm 0.0098	0.4180 \pm 0.0100	0.4111 \pm 0.0042	0.8471 \pm 0.0074

Table B11: Performance of NFS with its variants of NS equations ($n_t = 10$) on unseen meshes.

	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)	MAE($\times 10^{-3}$)	RMSE($\times 10^{-3}$)
	NS		NS		NS	
	$(n_s = 4096, n'_t = 10)$		$(n_s = 1024, n'_t = 20)$		$(n_s = 4096, n'_t = 40)$	
Flex + LN						
\mathbf{X}	0.9335 \pm 0.0011	1.3254 \pm 0.0012	1.8239 \pm 0.0012	2.5291 \pm 0.0008	3.2768 \pm 0.0026	4.3988 \pm 0.0009
$n'_s = 2n_s$	0.9731 \pm 0.0034	1.5042 \pm 0.0057	2.3530 \pm 0.0051	3.3320 \pm 0.0074	3.5439 \pm 0.0085	4.7904 \pm 0.0168
$n'_s = 3n_s$	1.1071 \pm 0.0021	1.5716 \pm 0.0038	2.5179 \pm 0.0089	3.5477 \pm 0.0125	3.6584 \pm 0.0180	4.8858 \pm 0.0246
$n'_s = 4n_s$	1.1015 \pm 0.0000	1.5627 \pm 0.0000	2.5919 \pm 0.0064	3.6526 \pm 0.0071	3.6608 \pm 0.0000	4.9521 \pm 0.0000
Gaus + LN						
\mathbf{X}	1.6341 \pm 0.0034	2.1992 \pm 0.0042	2.1976 \pm 0.0065	3.0219 \pm 0.0090	3.6422 \pm 0.0026	5.0097 \pm 0.0039
$n'_s = 2n_s$	2.8589 \pm 0.0062	4.0562 \pm 0.0126	3.7465 \pm 0.0041	5.1308 \pm 0.0097	3.9092 \pm 0.0041	5.2402 \pm 0.0075
$n'_s = 3n_s$	3.4513 \pm 0.0168	4.5199 \pm 0.0377	5.7712 \pm 0.0123	5.7137 \pm 0.0199	4.2102 \pm 0.0082	5.5057 \pm 0.0138
$n'_s = 4n_s$	3.4357 \pm 0.0000	4.7382 \pm 0.0000	5.5990 \pm 0.0066	5.5958 \pm 0.0049	4.2628 \pm 0.0000	5.7679 \pm 0.0000
Flex + LN						
\mathbf{X}	1.2138 \pm 0.0030	1.7293 \pm 0.0047	2.5119 \pm 0.0036	3.4923 \pm 0.0058	4.2083 \pm 0.0037	5.6761 \pm 0.0092
$n'_s = 2n_s$	1.4882 \pm 0.0146	2.1681 \pm 0.0300	7.0203 \pm 0.0203	10.6096 \pm 0.0345	5.8975 \pm 0.0060	8.7704 \pm 0.0189
$n'_s = 3n_s$	1.6384 \pm 0.0088	2.4130 \pm 0.0169	7.9177 \pm 0.0059	11.9825 \pm 0.0118	6.6622 \pm 0.0063	9.5874 \pm 0.0131
$n'_s = 4n_s$	1.6975 \pm 0.0000	2.5008 \pm 0.0000	7.1962 \pm 0.0101	10.8860 \pm 0.0098	6.6951 \pm 0.0000	9.6334 \pm 0.0000

The mesh-invariant evaluation on Burgers’ and KdV Equations of NFS are given in Table. B10. In Table. B10, when the spatial resolution is just 512, inference performance on unseen meshes deteriorates. This result also validates our conclusion (2) in the fourth paragraph in Sec. 3.3.

Besides, we give a full evaluation on mesh-invariance of NFS in NS equation, with its variants as a detailed results corresponding to Table. B11.

B.7 INTERPOLATION WITH OTHER VISION MIXERS

We conduct experiments on non-equispaced NS equations with the combination of our interpolation layers and other Vision Mixers to figure out if they can achieve comparable performance.

Table B12: Performance of different Vision Mixers combined with the interpolation layers in non-equispaced scenarios on NS equations ($n_t = 10$).

	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)
VIT	NS ($n_s = 4096, n'_t = 10$)		NS ($n_s = 1024, n'_t = 20$)		NS ($n_s = 4096, n'_t = 40$)	
\mathbf{X}	OOM	OOM	OOM	OOM	OOM	OOM
	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)
MLPMIXER	NS ($n_s = 4096, n'_t = 10$)		NS ($n_s = 1024, n'_t = 20$)		NS ($n_s = 4096, n'_t = 40$)	
\mathbf{X}	6.1854 \pm 0.0012	8.1556 \pm 0.0018	9.4593 \pm 0.0028	12.1316 \pm 0.0022	10.1862 \pm 0.0045	13.1548 \pm 0.0051
$n'_s = 2n_s$	8.1573 \pm 0.0126	11.2258 \pm 0.0147	12.0706 \pm 0.0132	14.9460 \pm 0.0159	10.6003 \pm 0.0127	13.6872 \pm 0.0238
$n'_s = 3n_s$	8.1952 \pm 0.0088	11.3840 \pm 0.0171	14.9910 \pm 0.0094	17.8415 \pm 0.0110	10.5633 \pm 0.0140	13.6394 \pm 0.0147
$n'_s = 4n_s$	8.7773 \pm 0.0000	11.3313 \pm 0.0000	14.9517 \pm 0.0125	17.7857 \pm 0.0199	10.5414 \pm 0.0000	13.6106 \pm 0.0000
	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)	MAE ($\times 10^{-3}$)	RMSE ($\times 10^{-3}$)
GFN	NS ($n_s = 4096, n'_t = 10$)		NS ($n_s = 1024, n'_t = 20$)		NS ($n_s = 4096, n'_t = 40$)	
\mathbf{X}	12.2373 \pm 0.0091	16.2902 \pm 0.0133	10.2768 \pm 0.0084	13.7852 \pm 0.0078	14.7765 \pm 0.0055	19.4872 \pm 0.0106
$n'_s = 2n_s$	13.7752 \pm 0.0164	18.3108 \pm 0.0181	17.7216 \pm 0.0225	24.1397 \pm 0.0371	15.9083 \pm 0.0235	21.0041 \pm 0.0256
$n'_s = 3n_s$	13.7054 \pm 0.0122	18.2192 \pm 0.0184	17.8238 \pm 0.0112	24.2783 \pm 0.0196	15.8986 \pm 0.0156	20.9943 \pm 0.0158
$n'_s = 4n_s$	13.7140 \pm 0.0000	18.2271 \pm 0.0000	17.8207 \pm 0.0105	24.2833 \pm 0.0141	15.8736 \pm 0.0000	20.9772 \pm 0.0000

B.8 COMPLEXITY COMPARISON

We here first give Table. B13 to show the complexity of time and memory of all the evaluated methods on NS ($r = 64, n_t = 10, n'_t = 40$).

Table B13: comparison on complexity of the evaluated methods

Type	Methods	Time/Epoch	Peak Memory	Parameter Number
Graph Spatio-Temporal Model	GCGRU	6'18"	8660MB	74945
	DCRNN	9'38"	11120MB	148673
	AGCRN	OOM	OOM	OOM
	MPPDE	10'54"	23333MB	622161
Vision Mixer	VIT	3'14"	32166MB	773217
	MLPMIXER	1'12"	4421MB	79749953
	GFN	48"	3296MB	1361729
	FNO	27"	3748MB	6299425
	PFNO	43"	3380MB	9742145
	NFS	2'02"	31938MB	37891937

Table B14: Detailed complexity of NFS

Interpolation on Resampled Points		
Neighbor Searching	Kernel Calculation	Weighted Summation
3522MB	3102MB	6884MB
Interpolation back on Original Points		
Neighbor Searching	Kernel Calculation	Weighted Summation
2506MB	2754MB	6884MB

It demonstrates that our method has comparable efficiency to Vision Mixers. For the graph spatial-temporal models, they suffer from the recurrent network structures and thus are extremely time-consuming while the parameter number is small, limiting their flexibility.

Time. However, once we compare the used time in PFNO and NFS, we will find that the interpolation layers are considerably time-consuming. Another module that cost time complexity is the normalization layer, as the original FNO does not include Layer-Norm in its architecture, but it is

stacked in PFNO. Theoretically, PFNO handles down-sampled grids in a low resolution, because of the patchwise embedding. However, it takes more time than FNO. Therefore, we conclude that the time complexity brought from Layer-Norm is very significant, but it is affordable because of the performance improvements.

Memory. Besides, the operation of searching for each spatial point’s neighborhood and calculating weighted summation in Eq. (9) and Eq. (10) are very memory-consuming. We test it on the same experiment, and give the memory usage of different models in forward process, as shown in Table. B14. The memory cost in backward process is 6902MB.

B.9 MORE VISUALIZATION

Here we provide more visualization results on the three equations. See Fig. B6, Fig. B7 and Fig. B8.

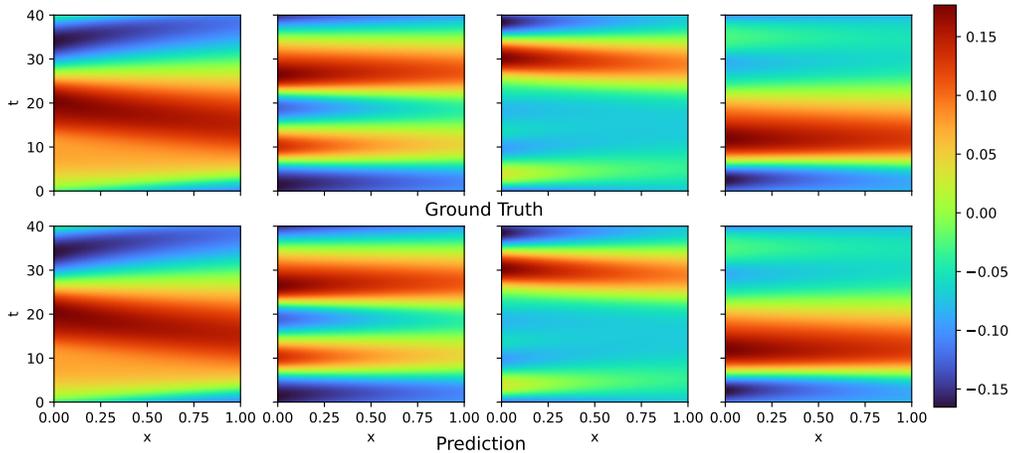


Figure B6: Visualization on equispaced Burgers’ equation.

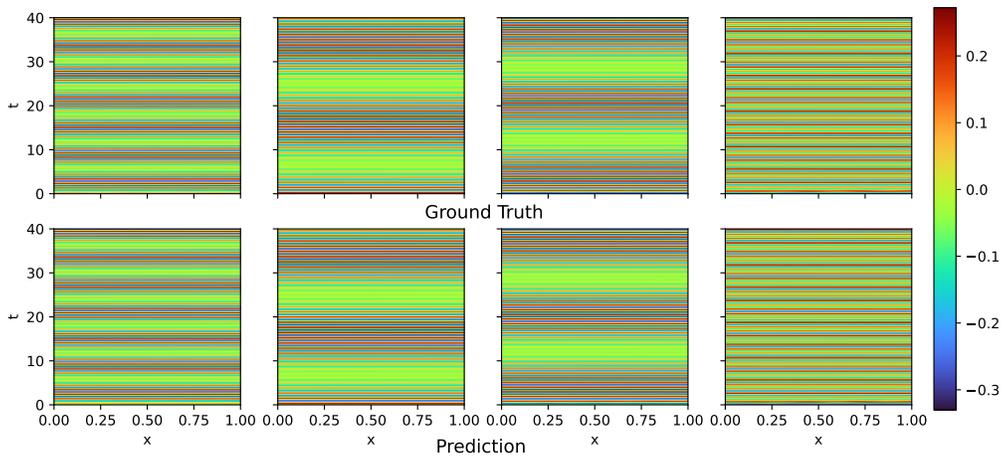


Figure B7: Visualization on equispaced KdV equation.

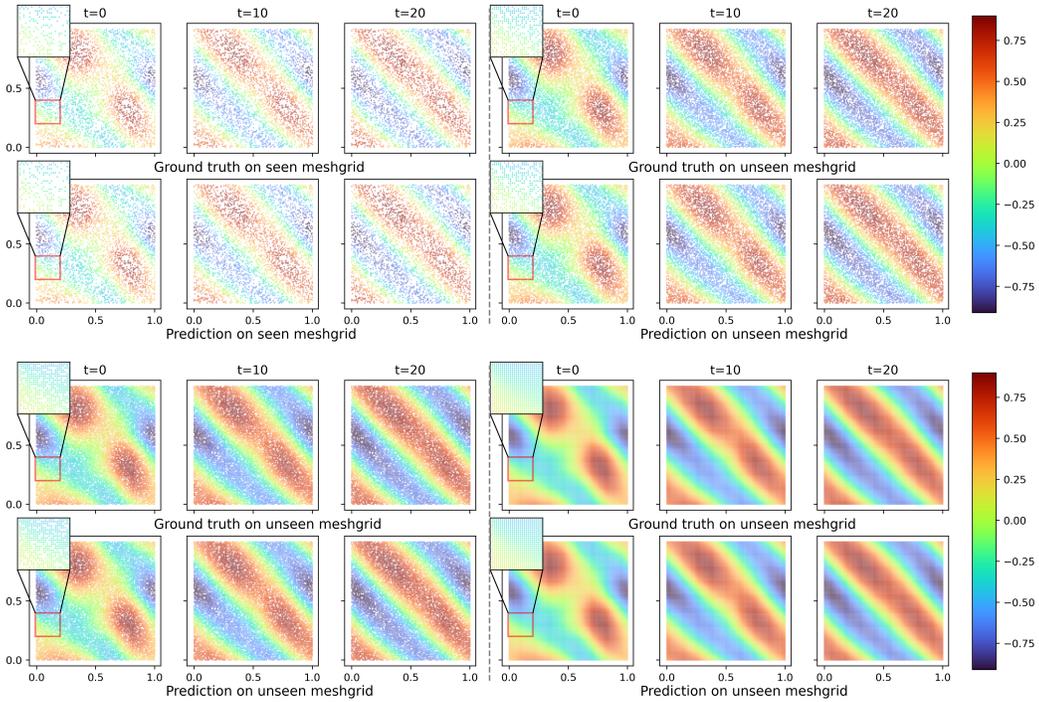


Figure B8: Visualization on non-equispaced NS equation: The training mesh ($n_s = 4096$ in upper-left) is different from the meshes in inference process ($n'_s = 8192$ in upper-right, $n'_s = 12288$ in lower-left and $n'_s = 16384$ in lower-right).

C EMPIRICAL OBSERVATION FOR THEOREM 3.1

In Sec. 3.2, Theorem 3.1 is proved to assure the expressivity of NFS. However, no further evidence gives the assurance of the convergence of the kernel interpolation. Here we conduct empirical study to give some clues.

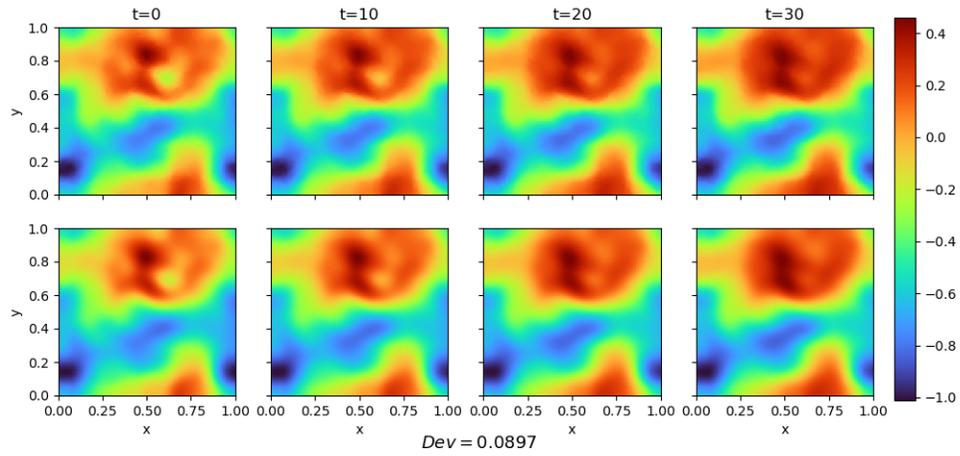
We conduct experiments on NS equation with $n_s = 4096$, $n_t = 10$, $n'_t = 40$. In a single trial, NFS is trained with fixed meshes. We repeated the trials 10 times with different meshes, and then give the one-v.s.-all deviations of the representation states calculated by

$$\text{Diff} = \frac{1}{90} \sum_{i \neq j, i, j=1}^{10} \frac{1}{m_s, n'_t} \left\| \frac{|H_i - H_j|}{|H_i| + |H_j|} \right\|_1,$$

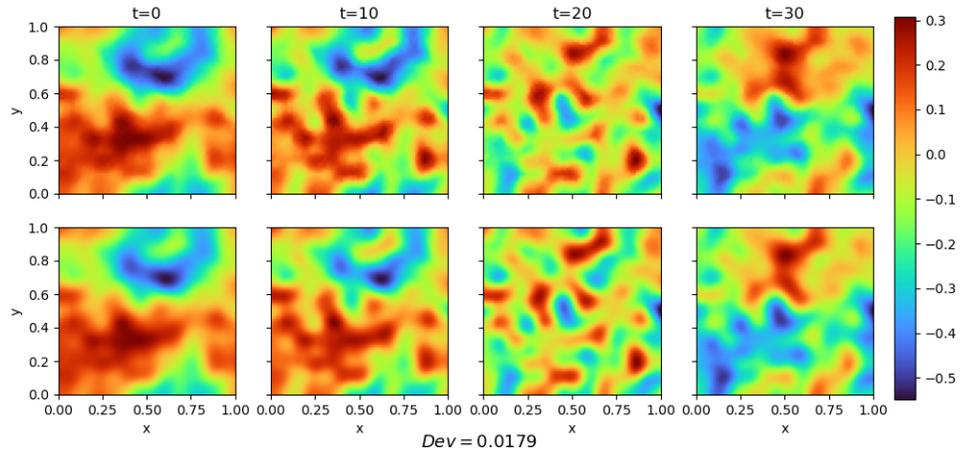
where H_i is the representation states of the shape $[\sqrt{m_s}, \sqrt{m_s}, n'_t]$, and $|\cdot|$ is the element-wise absolute value, and $\|\cdot\|_1$ is the 1-norm of the matrix. If the Diff is small in the beginning and end, it can be inferred that the interpolation kernel function converges to a similar mapping since the final predictions are close to ground truth in these experiments, and the inputs are sampled from the same instance of PDEs. We give the Diff before the first FNO and after the final of FNO layers in Table. C1. The small values indicate that the trained model usually has similar representation states. Figure. C1 and C2 give visualizations of representation states obtained by one instance of NS equation in two different trials. It indicates that the differences are getting smaller during the training.

Table C1: The defined Diff calculated by different epochs.

Epoch	Diff _{begin}	Diff _{end}
0	0.0676	0.0978
500	0.0102	0.0353



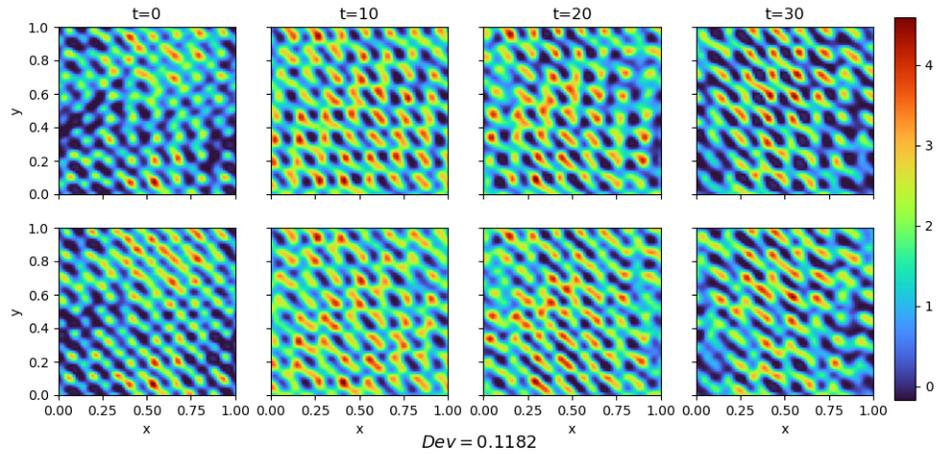
(a) Representation states at the beginning of FNO layers in two trials of Epoch 0



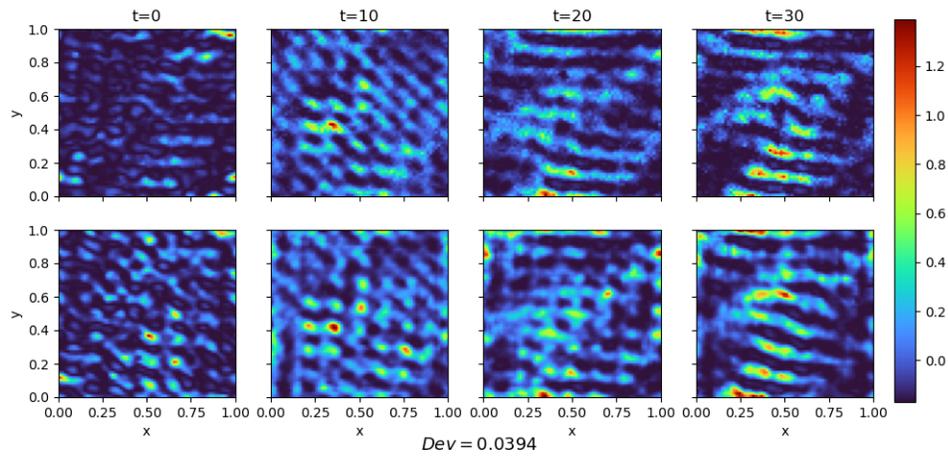
(b) Representation states at the beginning of FNO layers in two trials of Epoch 500

Figure C1: Visualization on different representation states at the beginning of FNO layers.

As a result, we present the one-v.s.-all differences of different epochs in the training process, to validate the convergence, as shown in Figure. C3.



(a) Representation states in the end of FNO layers in two trials of Epoch 0



(b) Representation states in the end of FNO layers in two trials of Epoch 500

Figure C2: Visualization on different representation states in the end of FNO layers.

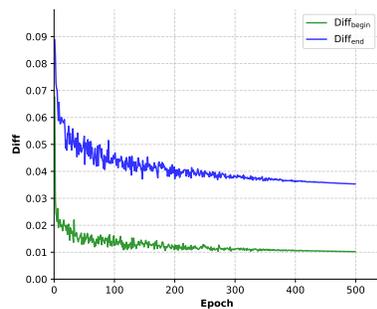


Figure C3: Convergence of Diff.