Computer Science
&
Data Science

Capstone Report - Fall 2023

On modifying Loss Function for Physics Informed Neural Operator(PINO) to approach Mathematics Nature in Fluid Mechanics

Kunyi (Mark) Ma
km5239@nyu.edu
Honors Mathematics, Data Science (Mathematics Concentration)

supervised by
Zhuang Su (zs2439@nyu.edu)

Preface

This project explores the mathematics possibility of Neural Operator Learning in Navier-Stokes Equations. We modified the original Data and PDE Loss functions in PINO using regularity theory from PDE Analysis, and found a Data Loss Norm that sees clearer turbulence flows. This promises more future collaborations between mathematics analysis and data science. The project is inspired by the author's self-recognition as a PDE Analyst in the field of pure mathematics, hence target audiences who are familiar with theoretical PDE while hoping to explore fields in Applied Mathematics.

Acknowledgements

I express my deepest gratitude to my advisor Dr. Zhuang Su, and Prof. Mathieu Lauriere for the discussions. I thank Mathematics Professors who led me to the study in Fluid Mechanics PDE Analysis, Prof. Vahagn Nersesyan and Prof. Vlad Vicol, and equally thank professor who've cultivated my PDE maturity up to this semester: Prof. Fanghua Lin, Prof. Chao Li, Prof. Pedro Antonio Santoro Salomão and Prof. Mac Huang. Finally, I wish to thank Yuejia Zhang, my girlfriend, whose unconditional love and emotional support helped me walk through the stressful semester.

Abstract

In this project, we explore a mathematics potential that fits PINO model better to the PDE in Neural Operator Learning. We care about 2D Incompressible Navier-Stokes, and leverage regularity theories from PDE Analysis to design 4 new Data and PDE Loss functions to the problem, $\mathcal{L}^s_{data}$, $\mathcal{L}^*_{pde}$, $\mathcal{L}^w_{data}$, *and* $\mathcal{L}^{w,t}_{data}$. *Under 3 sets of experiments,* $\mathcal{L}^s_{data}$ *designed assuming strong* H^1 *initial data stands out in predicting clearer turbulence flows compared with other methods and the original method. Hence this project introduces a promising idea that encodes proper PDE energy class into Data Loss function norm design, leading PINO into mathematics nature in Navier-Stokes to a greater extent.*

Keywords

Neural Operator; Deep Learning; Physics Informed Neural Operator; Physics Informed Neural Network; Fluid Mechanics; Navier-Stokes Equations; Sobolev Training

Contents

1 Introduction

1.1 context

Machine Learning with neural network has recently become promising in solving Partial Differential Equations (PDE), formulated as

$$\mathcal{L}_a u(x,t) = f(x,t) \quad x \in D,\ t \in [0,T]$$

The purpose was to learn the unknown function u with neural network, given operator $\mathcal{L}_a$, function f, and possible Initial/Boundary conditions. Note u is defined on each input $(x,t) \in D \times [0,T]$. Yet the previous work was subject to changes in the explicit form of $\mathcal{L}_a$, *i.e.*, for each new 'a' entering $\mathcal{L}_a$, the model always needs retraining. To tackle such a problem, Dr. Li and his collaborators proposed the idea of 'Neural Operator' which learns the operator that maps $\mathcal{L}$ from function spaces $a \in \mathcal{A}$ to $u \in \mathcal{U}$ [1], instead of the function $u(\cdot) = \mathcal{L}_a^{-1} f(\cdot) : D \times [0,T] \to \mathbb{R}$. They designed an iteration procedure with additional Kernel Integral Operator term

$$v_{t+1}(x) := \sigma\left(W v_t(x) + \int_D \left(K(a;\phi)\, v_t\right)(x-y)\, dy\right) \quad \forall\ x \in D$$

whose choice is well-known for the Fourier Integral Operator

$$\left(K(a;\phi)\, v_t\right)(x) = \mathcal{F}^{-1}\left(R_\phi \cdot \mathcal{F}(v_t)\right)(x) \quad \forall\ x \in D$$

with $\mathcal{F}$ as Fourier Transform and R_ϕ the parameter to learn [2]. Such a method, known as Fourier Neural Operator (FNO), can both solve the resolution issue for ∞-dimension function space and speed up computation with FFT.

In particular, they implemented their idea about FNO into a recently proposed method called Physics Informed Neural Operator (PINO). The PINO model combines two ideas in operator learning: using training model FNO, and minimizing loss function using Physics Informed Neural Networks (PINN)[3], a well-established design based on the physics constraints from PDE problem itself. The use of FNO is a supervised learning which requires input data, while minimizing loss based on PINN is an unsupervised learning that purely originates from the physics problem. One can think of PDE constraints as some 'true' guidance from mathematics that leads the model to the correct parameters.

Therefore, it's natural to construct 2 sets of loss functions to a well-formulated PINO problem. The first set $\mathcal{L}_{\text{pde}}$ serves as PDE loss function designed from the PDE constraint, which takes in source data $a \in \mathcal{A}$ and our model prediction, and fits the model to our PDE of interest. The second set $\mathcal{L}_{data}$ is Data loss, which takes in our model prediction and the true data that we have, fitting the model to the true data at hand. In concrete, for a given dynamical system

$$\begin{aligned} \frac{du}{dt} &= \mathcal{R}(u), && \text{in } D \times (0,\infty) \\ u &= g, && \text{in } \partial D \times (0,\infty) \\ u &= a && \text{in } \bar{D} \times \{0\} \end{aligned} \tag{1.1}$$

with $a = u(0) \in \mathcal{A}$ as the initial condition, g as known boundary condition, $\mathcal{R}$ a partial differential operator and the unknown $u \in \mathcal{U}$ for $\mathcal{A}$, $\mathcal{U}$ Banach spaces, the purpose is to learn the solution operator $\mathcal{G} : \mathcal{A} \to C([0,T]; \mathcal{U})$ defined by $a \mapsto u$. In PINO, they designed the **Operator PDE Loss Function** $\mathcal{L}_{pde}$ as

$$\mathcal{L}_{pde}(a, \mathcal{G}_\theta(a)) \equiv \mathcal{L}_{pde}(a, u_\theta) := \left\| \frac{d}{dt} u_\theta - \mathcal{R}(u_\theta) \right\|^2_{L^2((0,T];\mathcal{U}(D))} \tag{1.2}$$

$$+ \alpha \| u_\theta|_{\partial D} - g\|^2_{L^2((0,T];\mathcal{U}(\partial D))} + \beta \| u_\theta|_{t=0} - a\|^2_{\mathcal{U}(D)} \tag{1.3}$$

choosing $\mathcal{U}(D) = L^2(D)$ with hyperparameters α, β. Also, to minimize error between each output of neural operator $u_\theta = \mathcal{G}_\theta(a)$ and true value u, they have the original **Operator Data Loss Function** $\mathcal{L}_{data}$

$$\mathcal{L}_{\text{data}}(u, \mathcal{G}_\theta(a)) \equiv \mathcal{L}_{\text{data}}(u, u_\theta) := \|u - u_\theta\|^2_{\mathcal{U}} = \int_D |u(x) - u_\theta|^2 \mathrm{d}x \tag{1.4}$$

However, the original authors of PINO took it for granted that the natural norm $L^2((0,T]; L^2(D))$ they implemented in $\mathcal{L}_{pde}$ and $\mathcal{L}_{data}$ works for all PDEs they consider. But based on my own background from PDE analysis, norms for both loss functions $\mathcal{L}_{data}$ and $\mathcal{L}_{pde}$ could be, and should be carefully designed w.r.t. the specific PDE problem of interest due to proper regularity theories. In doing so, one may expect a faster and more stable convergence rate, as well as less error in $\mathcal{L}_{pde}$ loss while training.

1.2 objective

In this project, we care about the $2D$ periodic Navier-Stokes Equations for viscous and incompressible fluid in vorticity form, with supporting theories from [4], [5] [6] and [7]. We wish to justify the eligibility for modifying the loss functions $\mathcal{L}_{data}$ and $\mathcal{L}_{pde}$ to the problem of learning the operator $\mathcal{G}_{true}$ that maps initial condition to the full solution, *i.e.*, $\mathcal{G}_{true} : w_0 \mapsto w|_{[0,T]}$, whose original assumptions are $u \in C\left([0,T]; H^r\left(D; \mathbb{R}^2\right)\right)$ and $w_0 \in L^2\left(D; \mathbb{R}\right)$, $D = (0,l)^2, r > 0$

$$\begin{aligned} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + f(x) && x \in (0,l)^2,\ t \in (0,T] \\ \nabla \cdot u(x,t) &= 0 && x \in (0,l)^2,\ t \in (0,T] \\ w(x,0) &= w_0(x) && x \in (0,l)^2 \end{aligned}$$

where $w = \nabla \times u$ is vorticity, w_0 initial condition, $\nu \in \mathbb{R}_+$ the viscosity coefficient, and $f \in L^2_{\text{per}}((0,l)^2; \mathbb{R})$ the forcing function. In other words, we wish to train a model that predicts the behavior of $2D$ turbulence flow in a finite time future, given the present vorticity of the flow.

We've come up with 4 methods to modify loss functions in all. 3 of them lies in modifying the Data loss function $\mathcal{L}_{\text{data}}$ based on Uniqueness and Existence Theory from Navier-Stokes PDE Analysis; 1 modifies the PDE loss function $\mathcal{L}_{\text{pde}}$ leveraging *Sobolev Training* for PINN [5]. In particular, the new $\mathcal{L}_{\text{data}}$ are an innovation that incorporates information from PDE into the data loss design, by choosing proper norms matching regularity of the PDE.

A brief idea for the 4 designs are as follows (see detailed constructions in Section 3). Recall the original norm both to $\mathcal{L}_{data}$ $\mathcal{L}_{pde}$ are $L^2\left((0,T]; L^2(D)\right)$, *i.e.*, L^2 in both time and space:

1. [4] suggests replacing $\mathcal{L}_{data}$ norm $L^2(D)$ with results in Uniqueness and Existence theorem from Navier-Stokes assuming H^1 strong initial data. We use the norm for the energy class in *Theorem* 3.4 from [4] 'Global Existence and Uniqueness of strong solutions in 2D'.

2. [5] suggests replacing $\mathcal{L}_{pde}$ norm $L^2\left((0,T]; L^2(D)\right)$ with $L^2\left((0,T]; H^1(D)\right)$, *i.e.*, Sobolev norm H^1 in space, which incorporates the PDE's first-order derivative.

3. [4] also introduces the notion of Leray-Hopf Weak solution, which satisfies global existence in time assuming only L^2 initial data, but as a trade-off, sacrifices regularity. We try adopting such norm for the energy class in *Theorem* 4.2, 4.5 from [4] 'Existence and Uniqueness of Leray-Hopf weak solutions in 2D'.

4. [4] mentioned for Leray-Hopf Weak solutions, one shall require certain time derivative reg-

ularities in accordance to the energy class. So we try adding the time-regularity to $\mathcal{L}_{\text{data}}$ as suggested by *Theorem* 4.2, 4.5 from [4] 'Existence and Uniqueness of Leray-Hopf weak solutions in 2D'.

During the computation in H^1, we wish to try an equivalent Sobolev norm H^k, $k \in \mathbb{R}$ in Real Analysis [6], [7]

$$\|f\|_{H^k}^2 = \left\| \mathcal{F}^{-1} \left[\left(1 + |\xi|^2\right)^{\frac{k}{2}} \mathcal{F} f \right] \right\|_{L^2}^2 = \int_{\mathbb{R}^d} \left(1 + |\xi|^2\right)^k \left| \hat{f}(\xi) \right|^2 d\xi \tag{1.5}$$

which could be computed more efficiently by FNO in [2] via FFT and multiplication in Fourier Space. Moreover, this shall perform more stable than finite difference method due to invariance under resolutions.

1.3 contributions

We've coded and tested all Methods 1 - 4 with a Data Set as described in Sec 4.1. By comparing training data loss errors, testing data loss errors and model predictions, we conclude that a good choice of norm to capture mathematics nature in $2D$ Navier-Stokes is the choice of Strong Solution Norm $\mathcal{L}^s_{\text{data}}$ (3.2), which captures better physics phenomenon than the Original Method as in [3]. This finding and invention hence suggests it's important and promising to impose proper energy class (regularity) theory from PDE analysis into Data Loss Function design.

2 Related Work and Backgrounds

The literature review aims to cover the original work on PINO, and the theoretical background from 2D Navier-Stokes PDE that leads to the 4 norm designs to $\mathcal{L}_{pde}$, $\mathcal{L}_{data}$ I proposed.

2.1 Model - PINO

Traditionally, 'operators' are mathematical objects that map functions to other functions, which play a crucial role in PDE. As deep learning started making waves in a multitude of domains, the concept of a 'neural operator' emerged as an exciting synthesis of classical mathematical operators and neural networks.

The idea of Neural Operator started to mark its importance in the paper [1], which brings forward an innovative neural network architecture tailored for PDEs, defined as

$$\mathcal{G}_\theta := Q \circ (W_L + K_L) \circ \cdots \circ \sigma(W_1 + K_1) \circ P \tag{2.1}$$

where P and Q are pointwise operators,. The model stacks L layers of $\sigma(W_l + K_l)$ where W are pointwise linear operators, $K_l : \{D \to \mathbb{R}^{d_l}\} \to \{D \to \mathbb{R}^{d_{l+1}}\}$ are integral kernel operators and σ as activation function. The later paper [2] by Dr. Li *et al* on the Fourier Neural Operator (FNO), in particular, revised the iteration procedure

$$v_{t+1}(x) := \sigma\left(W v_t(x) + \int_D \kappa(x, y, a(x), a(y); \phi)\, v_t(y)\, dy\right) \quad \forall\, x \in D$$

by choosing the kernel operator as the Fourier Kernel, defined as

$$\kappa(x, y, a(x), a(y); \phi)\, v_t(y) \equiv (K(a; \phi)\, v_t)(x - y) := \mathcal{F}^{-1}(R_\phi \cdot \mathcal{F}(v_t))(x - y) \quad \forall\, x - y \in D$$

The Fourier kernels view convolution operation as multiplication in the Fourier space, and by setting a limit to frequency truncation in Fourier modes, they exhibit remarkable proficiency in capturing multi-scale features in many PDE problems. This characteristic not only bolsters computational efficiency but also ensures a degree of discretization invariance, making the model particularly adept at handling varied spatial-temporal data.

Meanwhile, the blending of classical physics with modern machine learning techniques, particularly led by PINN, has become an area of burgeoning research in solving PDEs. Building on

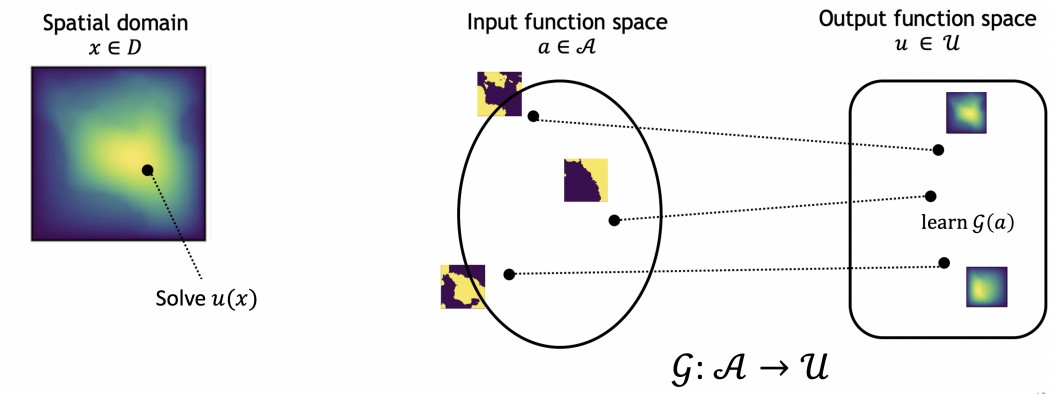

Figure 1: solve for one specific instance verse learn the entire solution operator
Left: numerical solvers and PINNs focus on solving one specific instance. Right: neural operators learn the solution operator for a family of equations. In PINO, for each iteration, we use Data loss to learn the operator for the Right, and then optimize the operator by minimizing *w.r.t.* PDE loss instance-wise for the Left.

foundational architectures in the FNO, Dr. Li *et al* in [3] introduced the Physics-Informed Neural Operator (PINO), whose brilliance lies in identifying that learning solution to PDEs falls into 2 paradigms: data-driven learning, and physics-informed optimization. As summarized in the introduction part from [3], PINO model aims to merge both paradigms by combining training data with a PDE loss function at a higher resolution, to ensure less degradation in accuracy on high-resolution tests if only low-resolution training data is provided. On one hand, it outperforms their old design in FNO, which cannot capture frequencies seen beyond training data, while on the other hand, it outperforms PINN because FNO computes gradients in Fourier space explicitly, which are more accurate than sampling-location-computations purely based on auto-differentiation *w.r.t.* PDE loss function. Figure 1 shall serve as a direct comparison between how PINN optimizing loss function solves equation pointwise, and how FNO as neural operator works upon training with data loss.

2.2 PDE - 2D Incompressible Navier-Stokes Cauchy Problem vorticity fomulation

The 'Navier-Stokes equations'(NS), a cornerstone in the study of fluid mechanics, capture the behavior of viscous fluid motion. In the realm of 'incompressible flows', the fluids are of constant density and prohibit volume changes, while in terms of 'vorticity formulation', solutions represent the fluid's local rotational effects [4]. In particular, the '2D case' is commonly favored in numerical methods as vorticity is of dimension 1 (while in 3D vorticity remains dimension 3), hence more numerically computable. Moreover, the associated 'Cauchy problem' focuses on the temporal evolution from a specified initial vorticity w_0, which makes it possible for operator learning by

viewing initial data w_0 as input, and its behavior over time $[0, T]$ governed by the equation, $w|_{[0,T]}$ as output [8]. Hence, the study on 2D Incompressible Navier-Stokes Cauchy Problem vorticity fomulation isn't only math problem, but also fits as perfect setting for Neural Operator Learning as proposed by Dr. Li *et al* [3].

We first present the PDE problem on 2D Navier-Stokes from [3]. Consider the 2D Navier-Stokes equation for a viscous, incompressible fluid in vorticity form on the unit torus, where $u \in C([0,T]; H^r_{\text{per}}((0,l)^2; \mathbb{R}^2)$ for any $r > 0$ is the velocity field, $w = \nabla \times u$ is the vorticity, $w_0 \in L^2_{\text{per}}((0,l)^2; \mathbb{R})$ is the initial vorticity, $\nu \in \mathbb{R}_+$ is the viscosity coefficient, and $f \in L^2_{\text{per}}((0,l)^2; \mathbb{R})$ is the forcing function. We want to learn the operator mapping the vorticity from the initial condition to the full solution $\mathcal{G}_{true} : w_0 \mapsto w|_{[0,T]}$.

$$\begin{aligned} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + f(x), && x \in (0,l)^2, t \in (0,T] \\ \nabla \cdot u(x,t) &= 0, && x \in (0,l)^2, t \in [0,T] \\ w(x,0) &= w_0(x), && x \in (0,l)^2 \end{aligned} \tag{2.2}$$

To prepare for our **norm design in Operator PDE Loss function** $\mathcal{L}_{pde}$, we mention some mathematical fact behind the particular NS setting, and how a computer computes it by 'solving velocity from vorticity'. Recall the 2D Biot-Savart's law in [4], which recovers velocity field $u = \begin{bmatrix} u_x \\ u_y \end{bmatrix}$ given vorticity $w \in \mathbb{R}$.

$$u(t) = -\nabla^\top (-\Delta)^{-1} w(t) \tag{2.3}$$

where the curl operator $-\nabla^\top := \begin{bmatrix} \partial_y \\ -\partial_x \end{bmatrix}$ in 2D. We call the scalar function ψ that solves the Poisson's equation $-\Delta\psi(t) = w(t)$ the stream function (which is scalar). Hence by inverting the Laplacian, we can represent $u(t) = \begin{bmatrix} \partial_y \psi \\ -\partial_x \psi \end{bmatrix}$ where $\psi = (-\Delta)^{-1} w$. Note that using Fourier Transform, differentiation and 'inverting differential operator' are equivalent to multiplying by powers of ik, where k is frequency from Fourier space and i is imaginary unit. Hence u can be computed easily from w upon using Fast Fourier Transform. Therefore given θ as parameter for the current neural operator, we're able to compute u_θ purely from using w_θ, hence the following

$$\mathcal{R}(w_\theta) := -u_\theta(x,t) \cdot \nabla w_\theta(x,t) + \nu \Delta w_\theta(x,t) + f(x) \tag{2.4}$$

where the forcing f and viscosity coefficient ν are initialized as part of the PDE problem. Therefore the original PDE loss function (1.2), using $\mathcal{G}_\theta$ from (2.1), in the 2D NS vorticity formulation case, reduces to

$$\mathcal{L}_{pde}\left(w_0, \mathcal{G}_\theta(w_0)\right) \equiv \mathcal{L}_{pde}\left(w_0, w_\theta\right) := \alpha \left\| \frac{d}{dt} w_\theta - \mathcal{R}(w_\theta) \right\|^2_{L^2\left((0,T];\, L^2\left((0,\ell)^2\right)\right)} + \beta \left\| w_\theta|_{t=0} - w_0 \right\|^2_{L^2\left((0,\ell)^2\right)} \tag{2.5}$$

where $\mathcal{R}(w_\theta)$ can be explicitly computed as above, and $\frac{d}{dt} w_\theta$ via finite difference method.

To prepare for our **norm design in Operator Data Loss function** $\mathcal{L}_{data}$, we similarly view it in the 2D NS setting as in (1.4), for w true solution and w_0 initial data

$$\mathcal{L}_{\text{data}}(w, \mathcal{G}_\theta(w_0)) \equiv \mathcal{L}_{\text{data}}(w, w_\theta) := \alpha \|w - w_\theta\|^2_{L^2([0,T);L^2((0,\ell)^2))} = \alpha \int_0^T \int_{(0,\ell)^2} |w - w_\theta|^2 \mathrm{d}x\mathrm{d}t \tag{2.6}$$

To introduce theories, we recall definition for Sobolev Spaces

$$H^k\left(D\right) := \left\{ f \in L^2\left(D\right) \mid \forall\ \alpha \in \mathbb{Z}^d_+ \ multi-index\ s.t.\ |\alpha| \leq k, \text{the distribution derivative } \partial^\alpha f \in L^2\left(D\right) \right\} \tag{2.7}$$

with multi-index $\alpha = (\alpha_1, \cdots, \alpha_d)$, of order $|\alpha| := \sum_{i=1}^d \alpha_i$ and $\partial^\alpha f \equiv \partial^{\alpha_1} \cdots \partial^{\alpha_d} f$. We also denote for simplicity

$$H^1_\sigma := L^2_\sigma \cap H^1 \quad where \quad L^2_\sigma := \left\{ u \in L^2 \mid \nabla \cdot u = 0 \text{ in distribution sense} \right\}$$

that captures 'incompressible' condition. Now we state the Theorems that we wish to follow from [4].

Theorem 2.1 (Global Existence and Uniqueness of strong solutions in 2D (Theorem 3.4 from [4])). *Fix $\nu > 0$ viscosity, and given $u_0 \in H^1_\sigma$ in 2D, there exists unique solution $u \in C\left([0,T)\,; H^1_\sigma\right) \cap L^2\left((0,T)\,; H^2_\sigma\right)$ that solves (2.2) with $f = 0$, where $T > 0$ is arbitrary.*

Theorem 2.2 (Existence and Uniqueness of Leray-Hopf weak solutions in 2D (Theorem 4.2, 4.5 from [4])). *Fix $\nu > 0$ viscosity, and given $u_0 \in L^2_\sigma$ in 2D, there exists unique Leray-Hopf weak solution $u \in L^\infty_{loc}\left(0, \infty; L^2_\sigma\right) \cap L^2_{loc}(0, \infty; \dot{H}^1_\sigma)$ whose time derivative has regularity $\partial_t u \in L^2_{loc}(0, \infty; \dot{H}^{-1})$ solving (2.2) in the sense of distributions with $f = 0$.*

Here norm in $\dot{H}^k$ means considering only highest order distributional derivative in H^k. And H^{-1} is dual space of H^1. We compute H^{-1} norm using Fourier Characterisation (1.5).

3 Norm Design - $\mathcal{L}^s_{data}$, $\mathcal{L}^*_{pde}$, $\mathcal{L}^w_{data}$, and $\mathcal{L}^{w,t}_{data}$

In this section, we explain in detail the 4 methods in loss function design, with possible algorithms included.

3.1 Method 1 - $\mathcal{L}^s_{data}$ Strong Solution Norm

We're proud to introduce a new set of Data Loss function of high originality in the domain of Data Science. They include information from the PDE solution's regularity, yet serve as pure Data Loss function, meaning the loss function themselves have less to do with the explicit PDE, but only takes important information from the solution space that the PDE generates. We can, of course, still define Data Loss functions using unusual norms without knowing the PDE. But it is because of the PDE that makes such particular choice of norm reasonable.

The paper by Dr. Li *at el* [3] did choose a good function class for expected solutions u and initial value w_0 to work with. Yet they seemed to stop there, and didn't quite make use of the regularity result that automatically establishes itself from Theorem 2.1. Under the similar assumptions compatible with the paper (the paper assumes $w_0 \in L^2$, hence $u_0 \in H^1$), velocity u automatically satisfies existence and uniqueness in the following class

$$u \in C\left([0,T)\,;H^1_\sigma\right) \cap L^2\left((0,T)\,;H^2_\sigma\right) \implies w \in C\left([0,T)\,;L^2_\sigma\right) \cap L^2\left((0,T)\,;H^1_\sigma\right) \tag{3.1}$$

meaning if w_θ were to converge *w.r.t.* the norm on RHS, then the limit is unique as the true solution w. Hence,

$$\mathcal{L}^s_{data}(w, \mathcal{G}_\theta(w_0)) \equiv \mathcal{L}^s_{\text{data}}(w, w_\theta) := \alpha \|w - w_\theta\|^2_{L^2([0,T);H^1((0,\ell)^2))} + \beta \|w - w_\theta\|^2_{L^\infty([0,T);L^2((0,\ell)^2))} \tag{3.2}$$

should be a more theoretical norm than the previous ordinary norm in L^2. Here L^∞ is simply defined as taking maximum, and H^1 the Sobolev norm.

When computing H^1 norm for each iteration, we try 2 methods: finite difference method using Sobel Operator Algorithm 1, and implementing the Fourier Differentiation Algorithm 2 making use of (1.5).

The finite difference method uses $dx = 1/64$ as compatible with spatial resolution for the input data dimension. For the Fourier Differentiation, we use Fast Fourier Transform and choose the lower half wave numbers in accordance with choice from [2], which truncates the finite number of

Algorithm 1 Compute Spatial Gradients using Sobel Operator

function COMPUTE_SPATIAL_GRADIENTS(w, dx)
    Define Sobel operators for x and y gradients:
    $sobel\_x \leftarrow \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \times \frac{1}{8dx}$
    $sobel\_y \leftarrow \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \times \frac{1}{8dx}$
    $grad\_x \leftarrow$ 2D convolution of w with $sobel\_x$
    $grad\_y \leftarrow$ 2D convolution of w with $sobel\_y$
    Return $grad\_x, grad\_y$
end function

Algorithm 2 Fourier Differentiation

function FOURIER_DIFFERENTIATION(w)
    $batchsize \leftarrow$ batchsize of w
    $nx \leftarrow$ size of x spatial dimension of w
    $ny \leftarrow$ size of y spatial dimension of w
    $w \leftarrow w$.reshape($batchsize, nx, ny$)
    $w_h \leftarrow$ 2D FFT of w over spatial domain
    $k\_max \leftarrow nx//2$
    $N \leftarrow nx$
    Generate k_x and k_y wavenumbers
    $wx\_h \leftarrow 1j \times k_x \times w_h$
    $wy\_h \leftarrow 1j \times k_y \times w_h$
    $wx \leftarrow$ Inverse FFT of $wx\_h[:, :, : k\_max + 1]$
    $wy \leftarrow$ Inverse FFT of $wy\_h[:, :, : k\_max + 1]$
    Return wx, wy
end function

Fourier modes for the FNO model. The feasibility is guaranteed by (1.5). Experimental results appear similar for both cases, so we stick to Fourier Differentiation in our following experiments that shall in theory expect more stability.

3.2 Method 2 - $\mathcal{L}^*_{pde}$ Sobolev PDE Loss function

As from [5], the main concept behind *Sobolev Training* is to minimize error between output and target function, and that between derivatives of output and target function, known as 'gradient matching'. By adding the first-order derivative term to the loss function, the neural network is penalized if its gradients deviate from the expected gradients (as given by the physical laws). This helps in stabilizing the training, which often results in faster convergence and better generalization, and is especially beneficial when the available data is sparse. In our case, it is equivalently

to rewriting our $\mathcal{L}_{pde}(w_0, \mathcal{G}_\theta(w_0))$ as

$$\mathcal{L}^*_{pde}\left(w_0, \mathcal{G}_\theta(w_0)\right) \equiv \mathcal{L}^*_{pde}\left(w_0, w_\theta\right) := \alpha \left\| \frac{d}{dt} w_\theta - \mathcal{R}(w_\theta) \right\|^2_{L^2\left((0,T];\, H^1\left((0,\ell)^2\right)\right)} + \beta \left\| w_\theta|_{t=0} - w_0 \right\|^2_{H^1\left((0,\ell)^2\right)} \tag{3.3}$$

that incorporates information about first derivative in space, with $\mathcal{R}$ defined in (2.4). Here we provide Algorithm 3 designed based on [3] that computes the object $D(w)$ (which later take difference with forcing f to obtain $\frac{d}{dt} w_\theta - \mathcal{R}(w_\theta)$), using finite difference method for time interval, yet Fourier Differentiation for spatial gradients. We plug the result into $L^2_t H^1_x$ norm as designed from Algorithm 2. Notice during Statement 14 to 16 in Algorithm 3, we've used Biot-Savart's law to recover u from w, as discussed in (2.3).

Algorithm 3 Finite Difference Method for Navier-Stokes Vorticity Formulation

procedure FDM_NS_VORTICITY($w, \nu, t\_interval$)
  Extract dimensions $batchsize, nx, ny, nt$ from w
  Reshape w to $(batchsize, nx, ny, nt)$
  Compute 2D FFT of w over x, y dimensions to get w_h
  Initialize wavenumbers k_x, k_y for x, y directions:
  $k\_max \leftarrow nx//2$
  $N \leftarrow nx$
  Generate wavenumber tensors k_x and k_y up to k_max modes in both spatial dimensions
  Compute negative Laplacian in Fourier space:
  $lap \leftarrow (k_x^2 + k_y^2)$
  Avoid division by zero at zero-frequency by setting $lap[0,0,0,0] \leftarrow 1.0$
  Compute stream function in Fourier space:
  $f_h \leftarrow w_h / lap$
  Compute velocity components in Fourier space:
  $ux_h \leftarrow 1j \cdot k_y \cdot f_h$
  $uy_h \leftarrow -1j \cdot k_x \cdot f_h$
  Compute gradient of vorticity in Fourier space:
  $wx_h \leftarrow 1j \cdot k_x \cdot w_h$
  $wy_h \leftarrow 1j \cdot k_y \cdot w_h$
  Compute Laplace of vorticity in Fourier space:
  $wlap_h \leftarrow -lap \cdot w_h$
  Convert Fourier space computations back to spatial domain:
  $ux \leftarrow$ Inverse FFT of ux_h
  $uy \leftarrow$ Inverse FFT of uy_h
  $wx \leftarrow$ Inverse FFT of wx_h
  $wy \leftarrow$ Inverse FFT of wy_h
  $wlap \leftarrow$ Inverse FFT of $wlap_h$
  Compute time step $dt \leftarrow t\_interval/(nt-1)$
  Compute time derivative of vorticity wt using finite difference:
  $wt \leftarrow (w[..., 2:] - w[..., :-2])/(2 \cdot dt)$
  Compute the Navier-Stokes term $D(w)$:
  $D(w) \leftarrow wt + (ux \cdot wx + uy \cdot wy - \nu \cdot wlap)[..., 1:-1]$
  Return $D(w)$
end procedure

Note that this is much stronger assumption than the previous $\mathcal{L}_{pde}$. Hence mimicing the proof provided in Theorem 7.4 from [5], which is originally carried using $1-D$ Burger's equations, we can show that

$$\|w(t) - \mathcal{G}_\theta(w_0)(t)\|_{L^\infty(0,T;L^2((0,\ell)^2))} \to 0 \quad as \quad \mathcal{L}^*_{pde}\left(w_0, \mathcal{G}_\theta(w_0)\right) \to 0$$

The essential ideas are integrating by parts, Hölder's Inequality and Grönwall's Inequality. Also, in the modification to $\mathcal{L}_{pde}$, we're only requiring regularity $u \in C\left([0,T]\,;\, H^1\left((0,\ell)^2;\, \mathbb{R}^2\right)\right)$ and $w_0 \in L^2\left((0,\ell)^2;\, \mathbb{R}\right)$, which matches the pre-assumed regularity class in the paper [3], meaning the provided dataset shall work well. Hence we're confident about obtaining good convergence result via this $\mathcal{L}^*_{pde}$. Finally, notice in our case, since the output dimension is essentially 1 due to vorticity formulation, we don't need to worry much about high-computation-costs for the derivative issues from [9] which have to use stochastic approximations for simplicity, but indeed introduced noises.

3.3 Method 3 & 4 - $\mathcal{L}^w_{data}$, $\mathcal{L}^{w,t}_{data}$ Weak Solution norm (and with time regularity)

Now compared to $\mathcal{L}^s_{data}$ as in Section 3.1, if we relax our notion of regularity for u a little bit, for example, viewing spatial derivative in the distribution sense, we can allow for u_0 initial velocity in L^2_σ. By result in Theorem 2.2, this gives Uniqueness and Existence to Leray-Hopf weak solution globally in $2D$, which are of lower regularity. The reason why we care about such notion is due to low regularity phenomenon observed in turbulence, known as Kolmogorov and Onsager theories of hydrodynamic turbulence. The ansatz of such theory is that, in the vanishing viscosity limit, solutions of Navier-Stokes Equations do not remain smooth uniformly w.r.t. ν, thus may only converge to distributional solutions of Euler's equations [10]. Hence in mathematics, weak solutions are in fact of more importance according to the physics nature. And in our dataset of interest, $\nu = 1/500$ is indeed small number, which makes sense to try using energy class to leray-Hopf weak solution. The full definition of Leray-Hopf weak solution is quite complicated, interested readers might find in [4]. Here we only need 2 facts about u

$$u \in L^\infty_{loc}\left(0,\infty; L^2_\sigma\right) \cap L^2_{loc}(0,\infty; \dot{H}^1_\sigma) \qquad \textit{the energy class} \tag{3.4}$$

$$\partial_t u \in L^2_{loc}(0,\infty; \dot{H}^{-1}) \qquad \textit{the time regularity} \tag{3.5}$$

There we shall leverage either the duality $w.r.t.$ $\dot{H}^1$ to compute $\dot{H}^{-1}$ or by Fourier Characterisation (1.5). And notice since u is no longer differentiable, we might need to again use Biot-Savart (2.3) to recover $u = \nabla^\top(\Delta)^{-1}w$ during computations in w (realized similarly in Algorithm 3), then compute the $\mathcal{L}^w_{data}$

$$\mathcal{L}^w_{data}(w, \mathcal{G}_\theta(w_0)) \equiv \mathcal{L}^w_{\text{data}}(w, w_\theta) := \alpha\|u-u_\theta\|^2_{L^2\left([0,T);\dot{H}^1((0,\ell)^2)\right)}+\beta\|u-u_\theta\|^2_{L^\infty([0,T);L^2((0,\ell)^2))} \quad (3.6)$$

and to add time derivative norm to $\mathcal{L}^{w,t}_{data}$

$$\mathcal{L}^{w,t}_{data}(w, \mathcal{G}_\theta(w_0)) \equiv \mathcal{L}^{w,t}_{\text{data}}(w, w_\theta) := \alpha\|u - u_\theta\|^2_{L^2\left([0,T);\dot{H}^1((0,\ell)^2)\right)} + \beta\|u - u_\theta\|^2_{L^\infty([0,T);L^2((0,\ell)^2))} \quad (3.7)$$

$$+ \gamma\|\partial_t u - \partial_t u_\theta\|^2_{L^2\left([0,T);\dot{H}^{-1}((0,\ell)^2)\right)} \quad (3.8)$$

We provide with Algorithm 4 that computes H^{-1} using the weighted L^2 norm in Fourier Space as in (1.5) at each fixed time step.

Algorithm 4 Compute Inverse H Norm

procedure H_INVERSE_NORM(f)
  Extract dimension nx from f
  Initialize wavenumbers k_x, k_y for x, y dimensions:
  $k\_max \leftarrow nx//2$
  Generate wavenumber tensors k_x and k_y
  Compute squared wave numbers $ksquared \leftarrow k_x^2 + k_y^2$
  Avoid division by zero at zero frequency by setting $ksquared[0, 0, 0] \leftarrow 1$
  Compute the Fourier transform of f:
  $F_f \leftarrow$ 2D FFT of f over x, y dimensions
  Compute the weighted L2 norm in Fourier space for each time step:
  $weighted\_L2\_per\_timestep \leftarrow \sum \left(\frac{|F_f|^2}{ksquared}\right)$ over x, y dimensions
  Subtract the zero-frequency component for each time step:
  $zero\_frequency\_norm \leftarrow |F_f[:, 0, 0]|^2$
  $weighted\_L2\_per\_timestep \leftarrow weighted\_L2\_per\_timestep - zero\_frequency\_norm$
  Return $weighted\_L2\_per\_timestep$
end procedure

4 Experiment and Results

With all 4 Norms $\mathcal{L}^s_{\text{data}}$ (3.2), $\mathcal{L}^*_{\text{pde}}$ (3.3), $\mathcal{L}^w_{\text{data}}$ (3.6) and $\mathcal{L}^{w,t}_{\text{data}}$ (3.8), we set up numerical experiments.

4.1 Data Explanation and Training Setup

We have a large Data Set of size 4000, each data having spatial resolution 64×64, and temporal resolution 64 (including both end points, so 65 time steps indexing from 0 to 64), with resolution based on unit cube of $[0, 1] \times [0, 1] \times [0, 1]$ (See Fig. 2 as example of a first training data at sample position 0 with time step 1). The 4000 data are taken from one whole simulation, where the 64th time step for one data is the 0th time step for the next one.

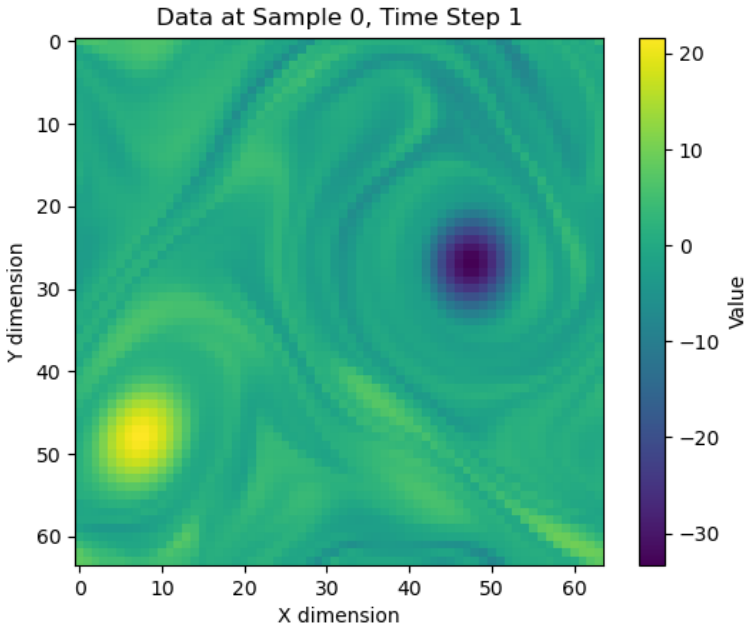

Figure 2: A training data sample at position 1 and time step 1

The training set samples 400 data from the beginning while the testing set samples the last 400. By setting t_interval =0.5 scale, we pre-process data by using data loader that sub-divides unit time interval into size of $[0, 0.5]$, $[0.25, 0.75]$, $[0.5, 1.0]$, $[0.75, 1.25]$ where the last interval takes $[0, 0.25]$ from the next data and attaches to $[0, 75, 1.0]$ for the previous one. Hence each data, with spatial resolution fixed, are modified into size of $(64, 64, 33)$. And while training, for each epoch, we train the model once, but test the errors on 4 consecutive data that were chopped off with overlapping time intervals. This procedure not only increases training data set size, but also incorporates some 'continuous' information one can convey to the loss functions, agreeing with

the dynamical system. For testing data, we use similar chopping procedure via dataloader, but only take 2 consecutive data to compute the testing errors and then take its average. We expect the testing data loss to fluctuate, even for the original model design with original loss functions (2.5), (2.6), so that we can make sharper comparisons.

Furthermore, for each epoch, we train model once, but test on 4 consecutive data, and hence call *loss.backwards*() with *optimizer.step*() 4 times. So for each epoch, each data iteration inherits loss function optimization from the previous data iteration, hence we manually push to model to 'correct' prediction for each consecutive 4 data as a whole. Now for this step, the original model uses $\mathcal{L}_{\text{data}}$ (2.6) and $\mathcal{L}_{\text{pde}}$ (2.5) to record $L_t^2L_x^2$ data loss and PDE loss. The loss they essentially updates are linear combinations of the 3 terms: $loss_{l2}$ from $\mathcal{L}_{\text{data}}$, $loss_f$ and $loss_{ic}$ from $\mathcal{L}_{\text{pde}}$, the latter denoting equation f loss and initial condition loss (recall we don't care about boundary data loss as we're dealing with periodic box $\mathbb{T}^2$). The linear coefficients sitting at the front are weights, as hyper-parameters, all set to 1. For consistency issues, when we try replacing $\mathcal{L}_{\text{data}}$ (2.6) and $\mathcal{L}_{\text{pde}}$ (2.5), we keep their corresponding weight summation the same. For instance when we do Strong solution norm $\mathcal{L}^s_{data}$ (3.2) which introduces both $L_t^2H_x^1$ and $L_t^\infty L_x^2$ norms, we take weight 4/5 for $L_t^2H_x^1$ and 1/5 for $L_t^\infty L_x^2$, which adds up to 1.

At each epoch, after 4 iterations of loss function optimization using data, which is semi-supervised, we add one more purely unsupervised iteration which only deals with PDE loss, using random initial data. We generate an online data loader using Gaussian Free Fields with chosen size (as suggested by Dr. Li *et al.* in [3]), use our model trained at this epoch to predict using this initial data, and plug in $\mathcal{L}_{\text{pde}}$ loss. Call $eqn_{loss}.backward()$ and *optimizer.step*() for 1 iteration. In particular, the random initial data and model predictions are of higher spatial resolution $(128, 128)$ which deals with degradation issues if only low-resolution training data is provided, as wrote in Section 2.1. This work is one of the major contributions to combine FNO and PINN.

4.2 Experimental Setup

We mainly conducted 3 sets of experiments.

- **Experiment 1**. We code all 4 loss functions $\mathcal{L}^s_{\text{data}}$ (3.2), $\mathcal{L}^*_{\text{pde}}$ (3.3), $\mathcal{L}^w_{\text{data}}$ (3.6) and $\mathcal{L}^{w,t}_{\text{data}}$ (3.8), and try different weight combinations with a short time epoch $= 160$, batch size $= 1$. We then fixed the weight combination for each norm for later computations. After getting familiar with the training loss behavior for each loss function, we compared finite difference

method with Fourier Differentiation using $\mathcal{L}^s_{\text{data}}$ and made the decision to implement Fourier Differentiation for future H^1 calculations.

- **Experiment 2**. We wish to study how the 4 original training losses provided in [3] behave using our newly designed 4 methods. The losses are:

  1. Data $L^2_t L^2_x$ error ($\mathcal{L}_{\text{data}}$ (2.6))
  2. Data f error (first term in $\mathcal{L}_{\text{pde}}$(2.5))
  3. Data IC L2 error (second term in $\mathcal{L}_{\text{pde}}$(2.5))
  4. Random IC Train equation loss (sum of both terms in $\mathcal{L}_{\text{pde}}$(2.5) with w_0 random initial data generated by Gaussian Free Field, see last paragraph of Section 4.1)

  In addition to the losses, we keep track of Time Cost Average, computed by: total time up to a given epoch divided by the given epoch. So

  $$\text{Time Cost Average} \times \text{the current epoch} = \text{Total time usage up to the current epoch}$$

  This reflects the efficiency of methods. In particular, we try to ensure a more stable convergence in general by the following 2 approaches: enlarging the batch size to 4; setting up a scheduler milestone that decreases the learning rate by half per 40 epochs.

- **Experiment 3**. We collect 3 test losses evaluated at test sample (the last 400 data), and keep the Random IC Train equation loss:

  1. TEST Data $L^2_t L^2_x$ error ($\mathcal{L}_{\text{data}}$ (2.6))
  2. TEST Data f error (first term in $\mathcal{L}_{\text{pde}}$(2.5))
  3. TEST Data IC L2 error (second term in $\mathcal{L}_{\text{pde}}$(2.5))
  4. Random IC Train equation loss

  This time we reduce batch size back to 1, and do not change base learning rate as epochs increase (since the suggested milestone in fact requires more epoch than our experiment epoch). We expect to see actual learning outcome of the models guided by our Methods. Finally we plot the testing data indexing at epoch 559 time step 32, and at epoch 599 time step 16, then compare the different actual predictions of our Methods.

4.3 Experiment 1

In philosophy, our original purpose of modifying loss functions is to approach the mathematics nature in Navier-Stokes Equations. Hence among the 4 original training loss, we particularly care about Data f error and Random IC Train equation loss, which are mainly information encoded within the PDE itself. The main comparison threshold for weights combinations and algorithm to calculate H^1 in the first Experiment thus lies in minimizing and witnessing a more stable convergence in Data f error and Random IC Train equation loss.

For finding the suitable weights combination, we provide an example for Method 1 $\mathcal{L}^s_{data}$ (3.2). At this stage, all weights are tested with the finite difference method $dx = 0.01$ to compute for H^1. In table 1, Data $L^2_t H^1_x$ error weight corresponds to α in $\mathcal{L}^s_{\text{data}}$ (3.2), Data $L^\infty_t L^2_x$ error weight to β in (3.2), while Data $L^2_t L^2_x$ error weight is α in $\mathcal{L}_{\text{data}}$ (2.6).

Index	Data $L^2_t H^1_x$ error weight	Data $L^\infty_t L^2_x$ error weight	Data $L^2_t L^2_x$ error weight
weight 1	$3/5$	$1/5$	$1/5$
weight 2	$1/5$	$1/5$	$3/5$
weight 3	$1/5$	$3/5$	$1/5$
weight 4	$4/5$	$1/5$	-

Table 1: Experiment 1 weight combinations for each data error.

Figure 3: Data f error and Random IC Train equation loss to Table 1

As shown in Fig 3, the engagement of original Data loss design seems to distract the convergence in both Data f error and Random IC Train equation loss as indicated by 'weight 4'. And the fact that 'weight 1' behaves the closest to 'weight 4' suggests the leading role of the first term in (3.2), *i.e.*, the $L^2_t H^1_x$ norm. Hence we pick the weight combination $4/5$ for α and $1/5$ for β in (3.2). The other norms are treated likewise. We provide with the following weight choices Table 2 for Data Loss and Table 3 for PDE Loss to are our Methods 1 - 4 fixed for future experiments. All weights correspond to certain choice of hyper-parameters α, β, γ from (2.6), (2.5), (3.2), (3.3),

(3.6), (3.8). In Table 3, f stands for α in either (2.5) or (3.3), IC stands for β in either (2.5) or (3.3).

Method	$L_t^2 L_x^2$	$L_t^2 H_x^1$	$L_t^\infty L_x^2$	weak $L_t^2 \dot{H}_x^1$	weak $L_t^\infty L_x^2$	weak $L_t^2 \dot{H}_x^{-1}$
Method 1 $\mathcal{L}^s_{\text{data}}$	-	4/5	1/5	-	-	-
Method 2 $\mathcal{L}^*_{\text{pde}}$	1	-	-	-	-	-
Method 3 $\mathcal{L}^w_{\text{data}}$	-	-	-	4/5	1/5	-
Method 4 $\mathcal{L}^{w,t}_{\text{data}}$	-	-	-	3/5	1/5	1/5

Table 2: Data error $\mathcal{L}_{\text{data}}$ weights for Methods 1-4

Method	f $L_t^2 L_x^2$	IC L_x^2	f $L_t^2 H_x^1$	IC H_x^1
Method 1 $\mathcal{L}^s_{\text{data}}$	1	1	-	-
Method 2 $\mathcal{L}^*_{\text{pde}}$	-	-	1	1
Method 3 $\mathcal{L}^w_{\text{data}}$	1	1	-	-
Method 4 $\mathcal{L}^{w,t}_{\text{data}}$	1	1	-	-

Table 3: PDE error $\mathcal{L}_{\text{pde}}$ weights for Methods 1-4

With fixed choices for weight combinations, we select a method to compute for spatial derivatives H_x^1. We conduct an experiment using Finite Difference Method Algorithm 1 $dx = 1/100$, $dx = 1/64$ and Fourier Differentiation Algorithm 2, tracing Data f error and Random IC Train equation losses, see Fig 4. The 'fourier' one indicated in both errors appear less than the results for finite difference methods, whereas for different resolutions dx, we nearly see no difference. Hence we propose keeping Fourier Differentiation as our tool to compute spatial H_x^1 in later experiments.

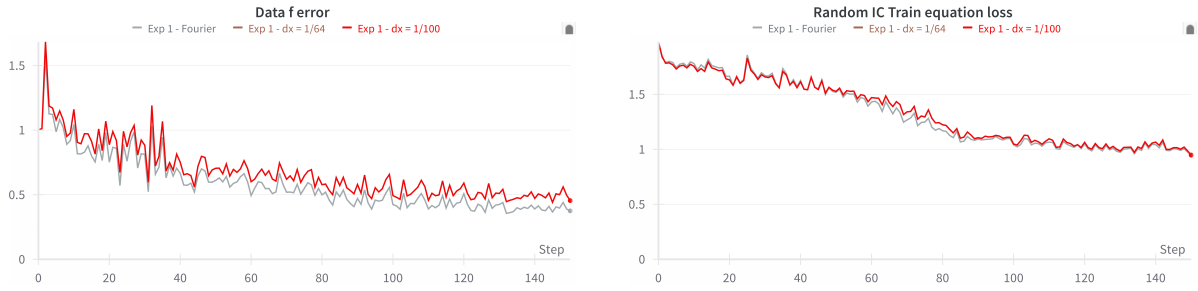

Figure 4: Data f error and Random IC Train equation loss to compute H_x^1

4.4 Experiment 2

We begin by justifying that choosing batch size = 4 indeed makes convergence more stable. As typical example, we still used Method 1 whose spatial gradient computation incorporates great

fluctuations. This time we not only choose as usual Data f error, but also use Data train loss, which is sum of Data f, Data IC and Data Loss (in $\mathcal{L}^s_{\text{data}}$ case, $4/5\ L_t^2 H_x^1 + 1/5\ L_t^\infty L_x^2$). We observe that in both errors, a batch of size 4 outperforms batch size 1 in stability, as in Fig 5.

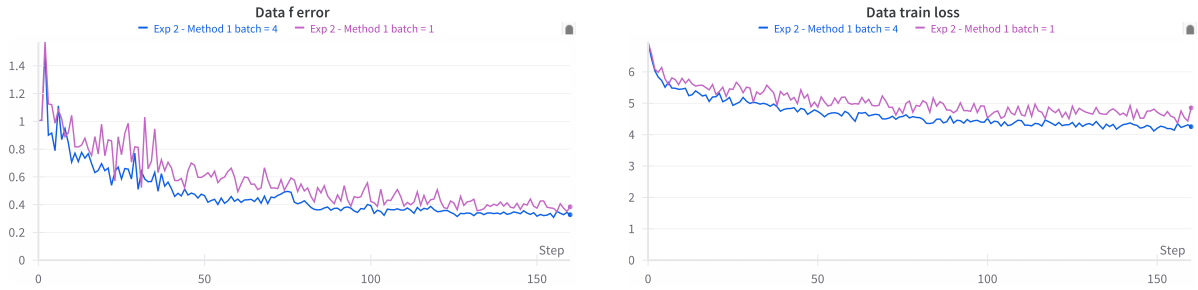

Figure 5: Data f error and Data training loss for batch size = 1, 4

Now we present the training data losses along with Time Cost Average for the Original Method, and Methods 1 - 4, see Fig 6, 7, 8, 9, 10. Notice we're comparing the training loss, instead of testing loss, hence the aim lies in comparing how well, how fast, and how stable our methods fit the training data. We may extract the following information via comparison between these 5 figures:

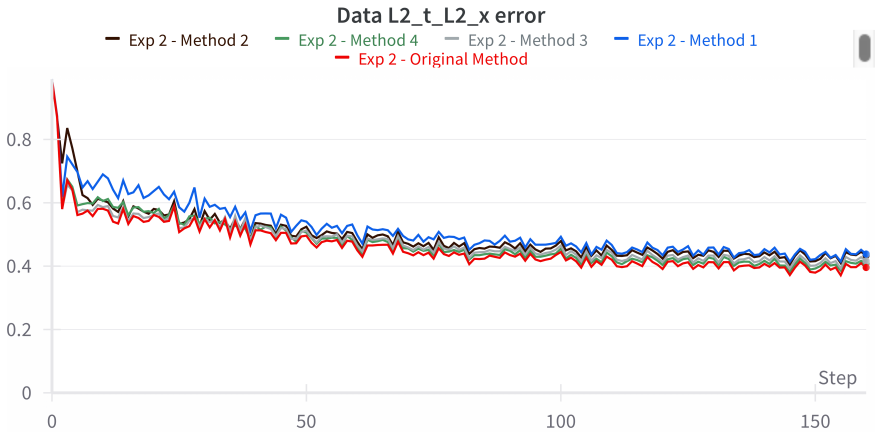

Figure 6: Training Data $L_t^2 L_x^2$ error

- From Data $L_t^2 L_x^2$ error Fig 6 and Data IC L2 error Fig 8, we clearly observe the trend that Method 1 $\mathcal{L}^s_{\text{data}}$ converges the poorest while the Original Method (2.5), (2.6) converges the best. However, on the other hand, from Data f error Fig 7 and Random IC Train equation loss Fig 9, we observe the opposite trend that Method 1 $\mathcal{L}^s_{\text{data}}$ converges the best while the Original Method converges the poorest. This to a great extent matches our hypothesis that, making use of Uniqueness and Existence Theorem assuming H^1 initial data u_0, our energy

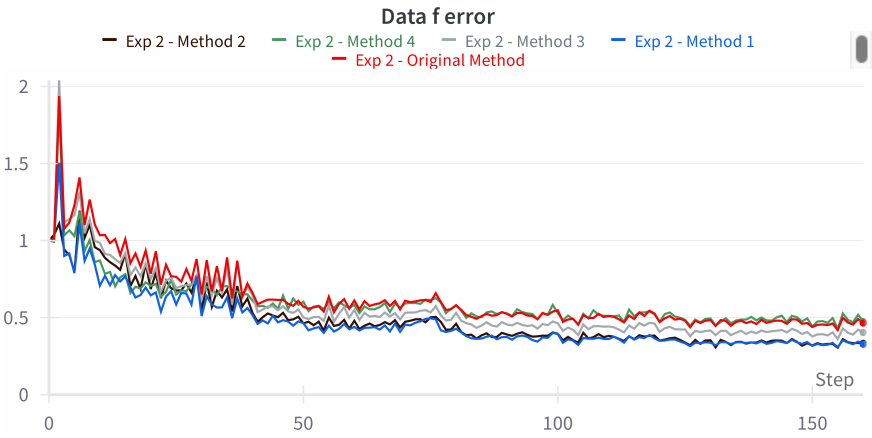

Figure 7: Training Data f error

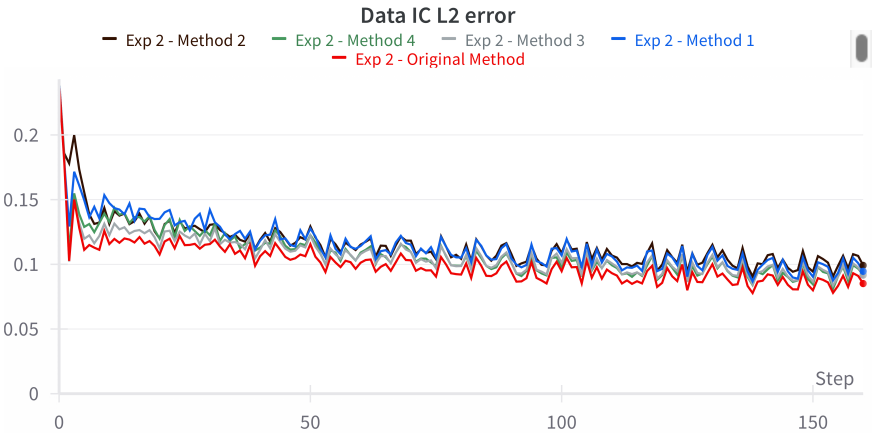

Figure 8: Training Data IC L2 error

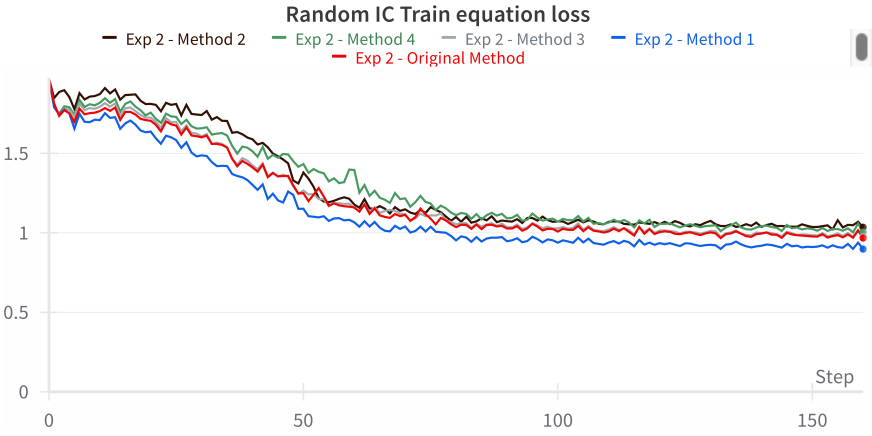

Figure 9: Random IC Train equation loss

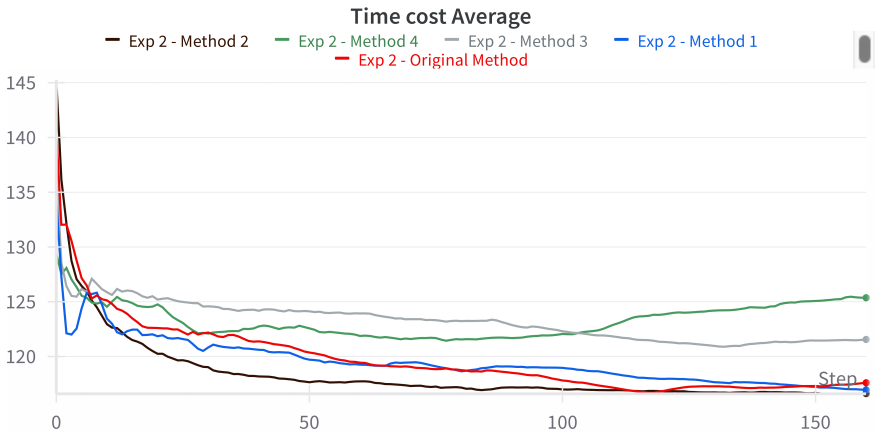

Figure 10: Training Time Cost Average

class (3.1) shall be the correct function space to minimize norm in. This piece of information inherited from PDE indeed ensures a better PDE error convergence, hence sacrificing Pure Data error, even for initial condition.

- We compare the behavior of Method 1 $\mathcal{L}^s_{\text{data}}$ and Method 2 $\mathcal{L}^*_{\text{pde}}$ in Fig 6 to 8, and observe that their convergence behavior matches the most as epochs increase. However, in Random IC Train equation loss Fig 9, we notice a great distinction, where Method 1 $\mathcal{L}^s_{\text{data}}$ way outperforms Method 2. The phenomenon suggest that Method 1 $\mathcal{L}^s_{\text{data}}$ is better at dealing with random initial data, hence learned more information about the PDE. On the other hand, in fact, Sobolev Training with $\mathcal{L}^*_{\text{pde}}$ does not give the model any more information about the PDE itself, even though the first order gradient matching in the loss function behaves well in Fig 6 to 8. It is their distinction in Fig 9 that justifies: it is the matter of 'lying in the correct energy class' that ensures good convergence in Method 1 $\mathcal{L}^s_{\text{data}}$, not necessarily training with gradient matching. Also, due to direct gradient matching for both Methods 1, and 2, Time Cost Average Fig 10 shows they cost the least training time.

- Whether our Method learns information well in the PDE is essentially reflected in its behavior on Random IC Train equation loss Fig 9. Notice except for Method 1, all other methods seem to behave similar, even worse in terms of convergence. We focus on Method 3 $\mathcal{L}^w_{\text{data}}$ and Method 4 $\mathcal{L}^{w,t}_{\text{data}}$, and notice they behave similarly on Fig 9 in terms of 'not catching information from PDE'. For Fig 6 and Fig 8, they lie somewhere in between the Original Method and Method 1, indicating a certain improvement(compared to Method 1) in capturing actual data loss yet not to the level of original Method. Method 3 and 4 did

show a distinction in Fig 7, where Method 4 $\mathcal{L}^{w,t}_{\text{data}}$ behaves worse in capturing data f error, even up to the level of the Original Method. This does reflect the original purpose of defining Weak Norms, which are to sacrifice regularity for integrability, so they should behave less sharp and more stable. However in the training losses, we did not see an enhancement in fitting to the PDE, so we question the working eligibility of Method 3 and 4 for now. And do observe, due to applying Biot-Savart that translated back from vorticity to velocity at each epoch, the Time Cost Averages behave terrible for both methods.

4.5 Experiment 3

We reduce batch size = 1 and keep the learning rate fixed for the first 600 epochs. This corresponds to the original setup in [3]. Strictly following weight combinations in Table 2 and Table 3, we obtain Figure 11, 12, 13, 14 and 15.

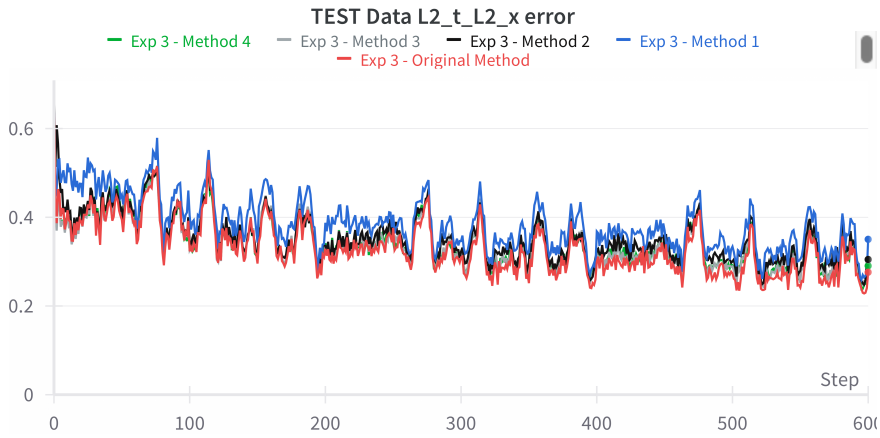

Figure 11: TEST Data $L^2_t L^2_x$ error

We add a few more observations to the loss functions:

- As shown in Random IC Train equation loss Fig 14, after around 350-400 epochs, expect for Method 2, all other Methods term to converge to the same behavior, with Method 1 performing a bit better, but Methods 3 and 4 are not 'worse', turning a negative answer to what was suggested in the previous experiment. Methods 3 and 4 indeed behave more stable and lie between the Original Method and Method 1 in TEST Data $L^2_t L^2_x$ error Fig 11 and TEST Data IC L2 error Fig 13, indicating an adaptation from PDE theorems to the supervised learning Model, and indeed preserve learning outcomes on the PDE itself. However, both Methods 3 and 4, in general, perform not as good as the Original Method,

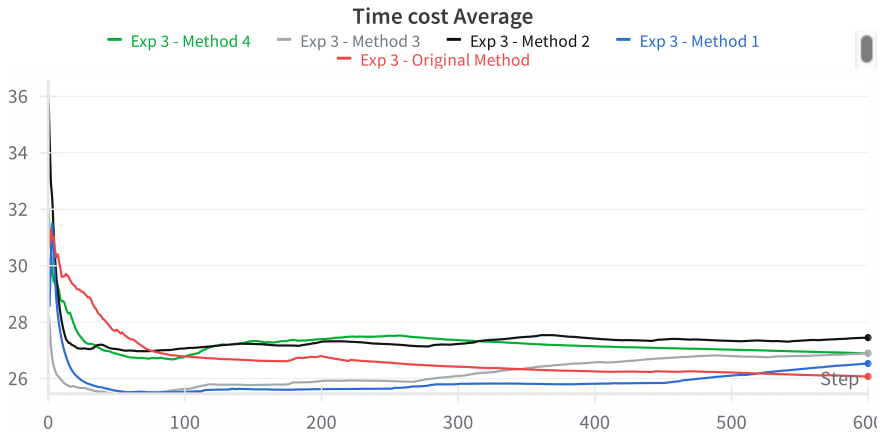

Figure 15: Time Cost Average

while remaining high in Time Cost Average Fig 15. Hence they themselves are not suitable loss functions to train a PINO model with.

- Meanwhile, Method 2 'unexpectedly' remains high in Random IC Train equation loss Fig 14, indicating Sobolev Training modification to the equation itself does not learn much information about the PDE itself. Combine with its poor Time cost Average behavior in Fig 15, we conclude as well that Method 2 itself is not suitable loss functions to train a PINO model with.

- Finally, for Method 1, we observe similar performance on Data f error Fig 12 and Random IC Train equation loss Fig 14 compared with its behavior on training losses in Experiment 2. Among the 4 Methods we've studied, Strong Solution norm $\mathcal{L}^s_{\text{pde}}$ (3.2) captured the most on PDE information, outperforming the Original Method, and deserves a further investigation.

Now we plot the actual prediction for Original Method and Method 1 at test sample 559 time step 32, see Figure 16. Due to a relatively early stage in training, we see both the Original Method and Method 1 does not perform well in learning the true data, yet there are differences lying between the Model Predictions for Original Method and Method 1. In particular, our Method 1 sees clearer the following curves compared with Original Method

- x-dimension around 20 and y-dimension 50-60

- x-dimension 50-60 and y-dimension 10-20

- the boundary between 2 curves at x-dimension 20-30 and y-dimension 20-30

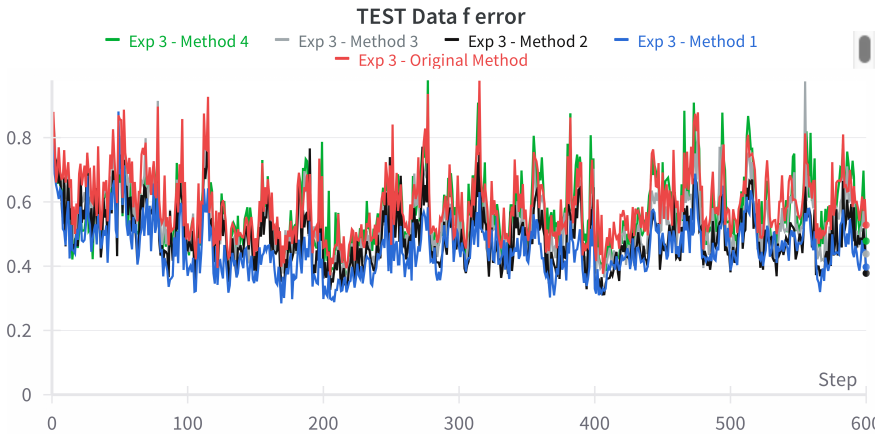

Figure 12: TEST Data f error

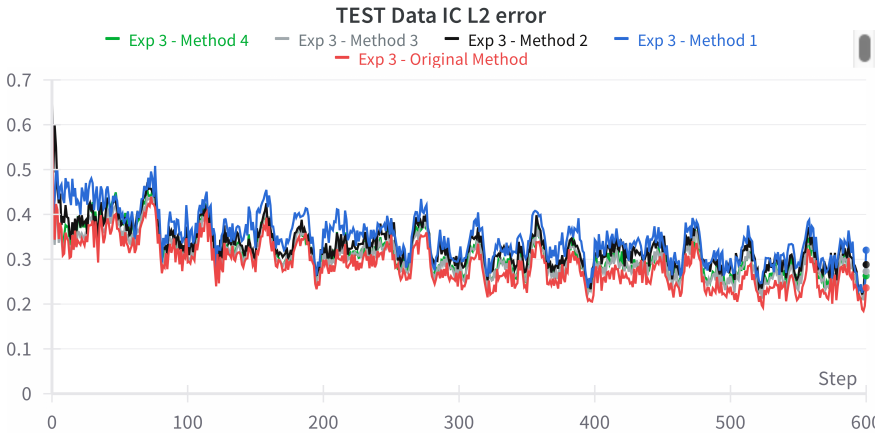

Figure 13: TEST Data IC L2 error

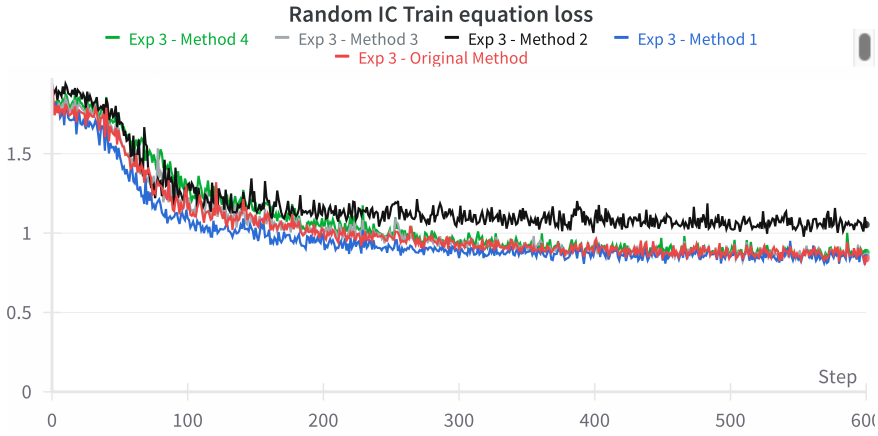

Figure 14: Random IC Train equation loss

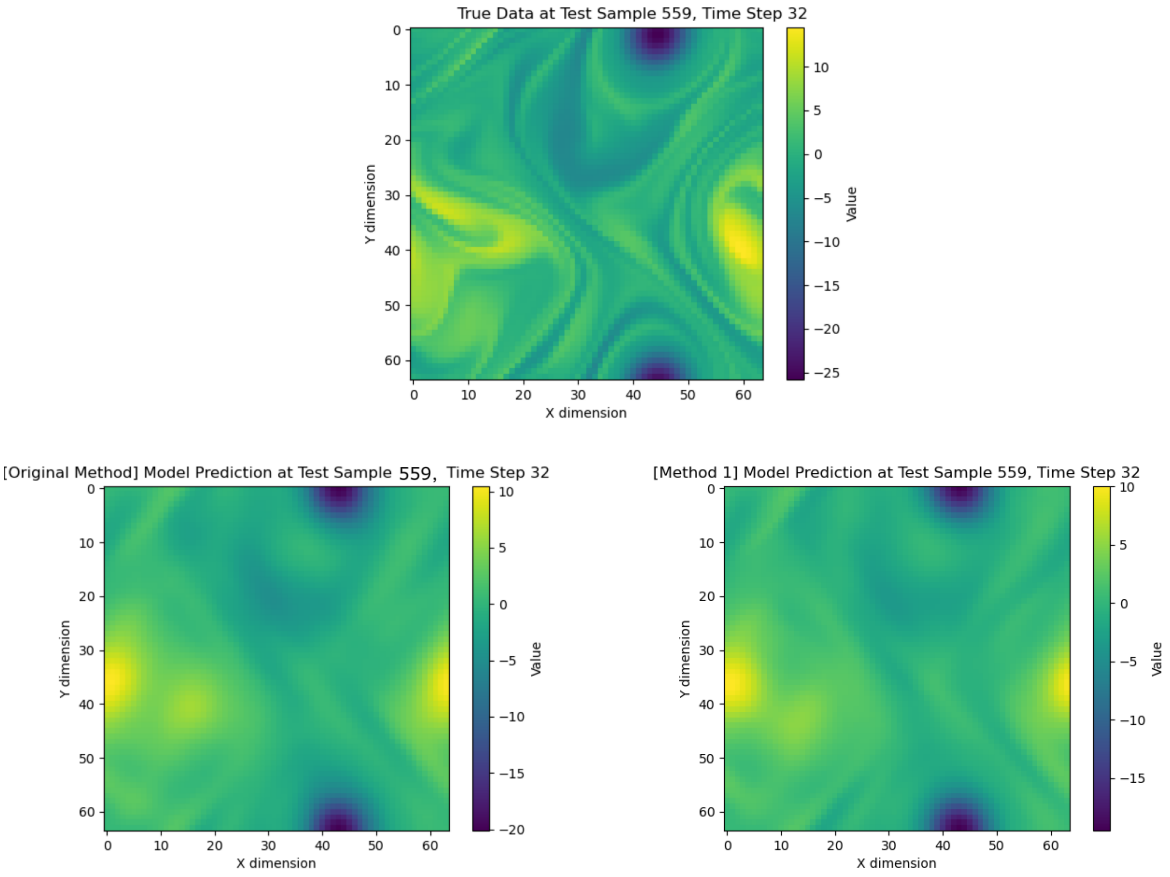

Figure 16: Test Sample 559 at time step 32 (True Data, Original Method, Method 1)

Further demonstrating the difference, we include another set at Test Sample 599, but at an earlier time step 16, see Fig 17. An earlier time step prediction, in principle, shall be easier to capture flows. Indeed, we again make observations that our Method 1 sees clearer the following curves compared with Original Method

- x-dimension 20-40, y-dimension 0-20
- x-dimension 40-60, y-dimension 30-50
- the boundary between 2 curves at x-dimension 0-20, y-dimension 30-40

However, also note that the Original Method sees a sharper color contrast in Fig 17, which matches more with the True Data, meaning the overall 'values' are predicted better. On the other hand, Method 1 overall appears 'darker', but did see 'thinner' and clearer curves that matches more with the True Data. This observation agrees with our loss function trackings in both training data and testing data, and further justifies our hypothesis that: Making use of Strong solution norm, *i.e.*, inheriting more information from PDE to the data loss function design, really does give better predictions in data that captures physical movement in turbulence.

For matters of completion, we post model predictions on same data Test Sample 559, time step 32, and on Test Sample 599, time step 16, using Methods 2, 3, 4. One can be careful and observe that the model predictions are in general blurred and doe not give good predictions as Method 1. See Fig 18 and Fig 19 for comparison over all methods.

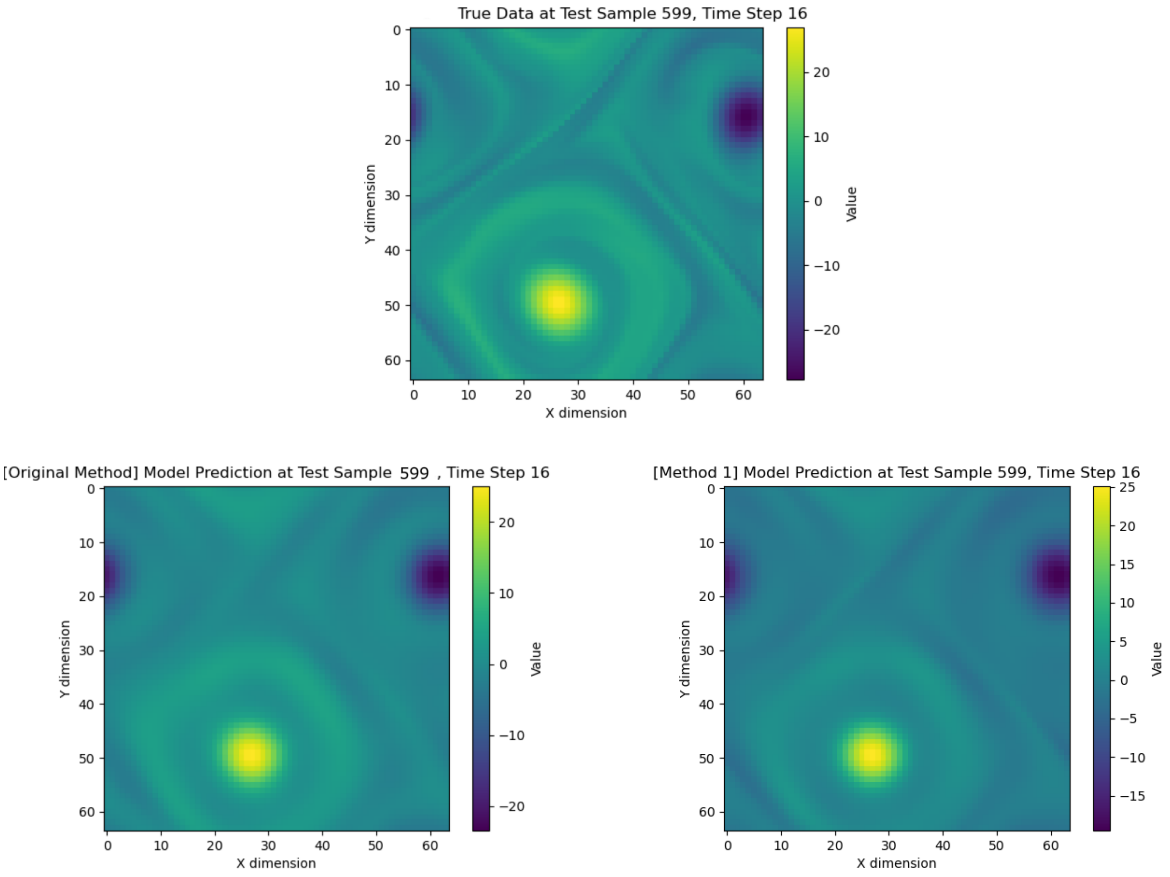

Figure 17: Test Sample 599 at time step 16 (True Data, Original Method, Method 1)

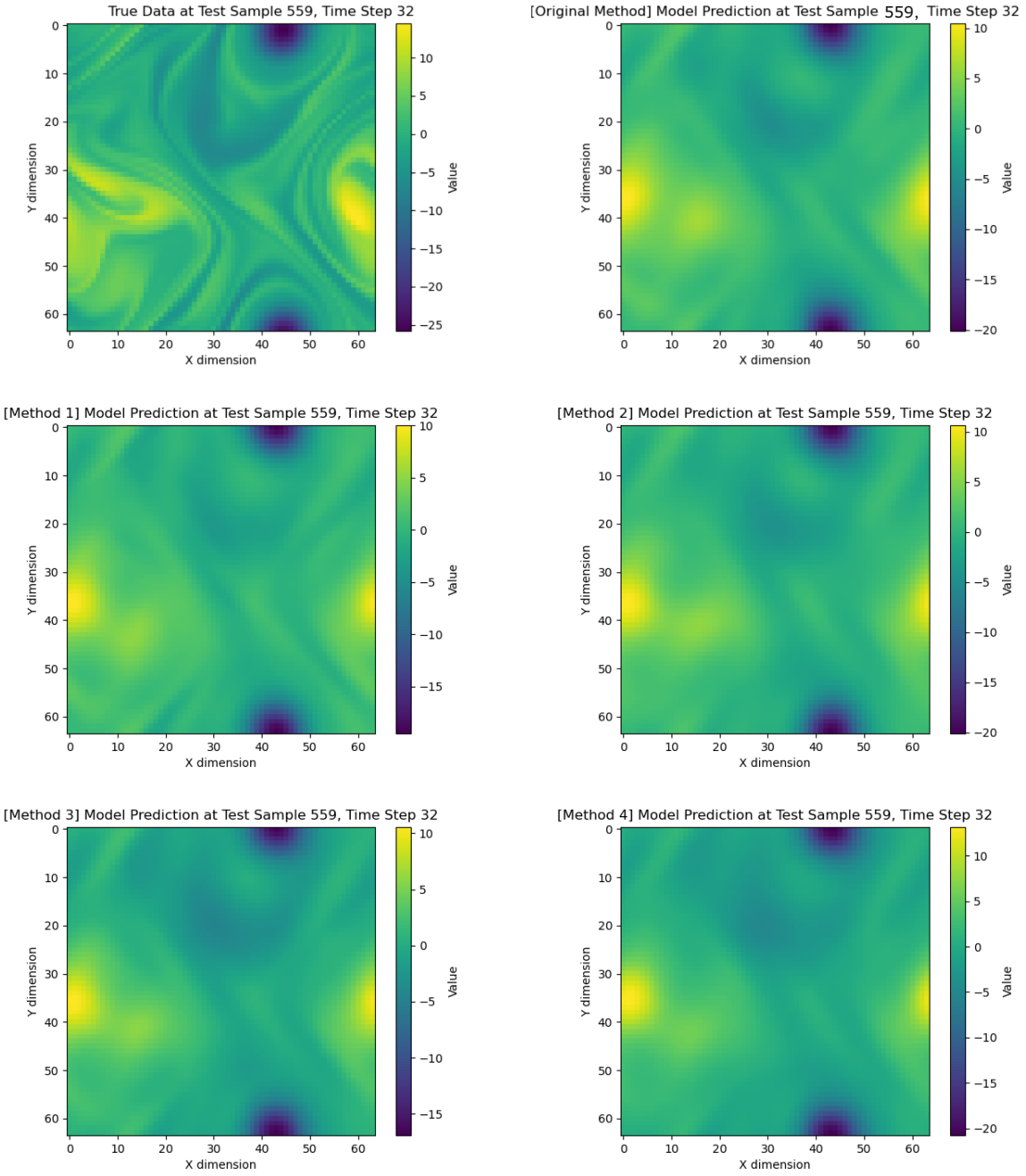

Figure 18: Test Sample 559 at time step 32 (True Data, Original Method, Methods 1-4)

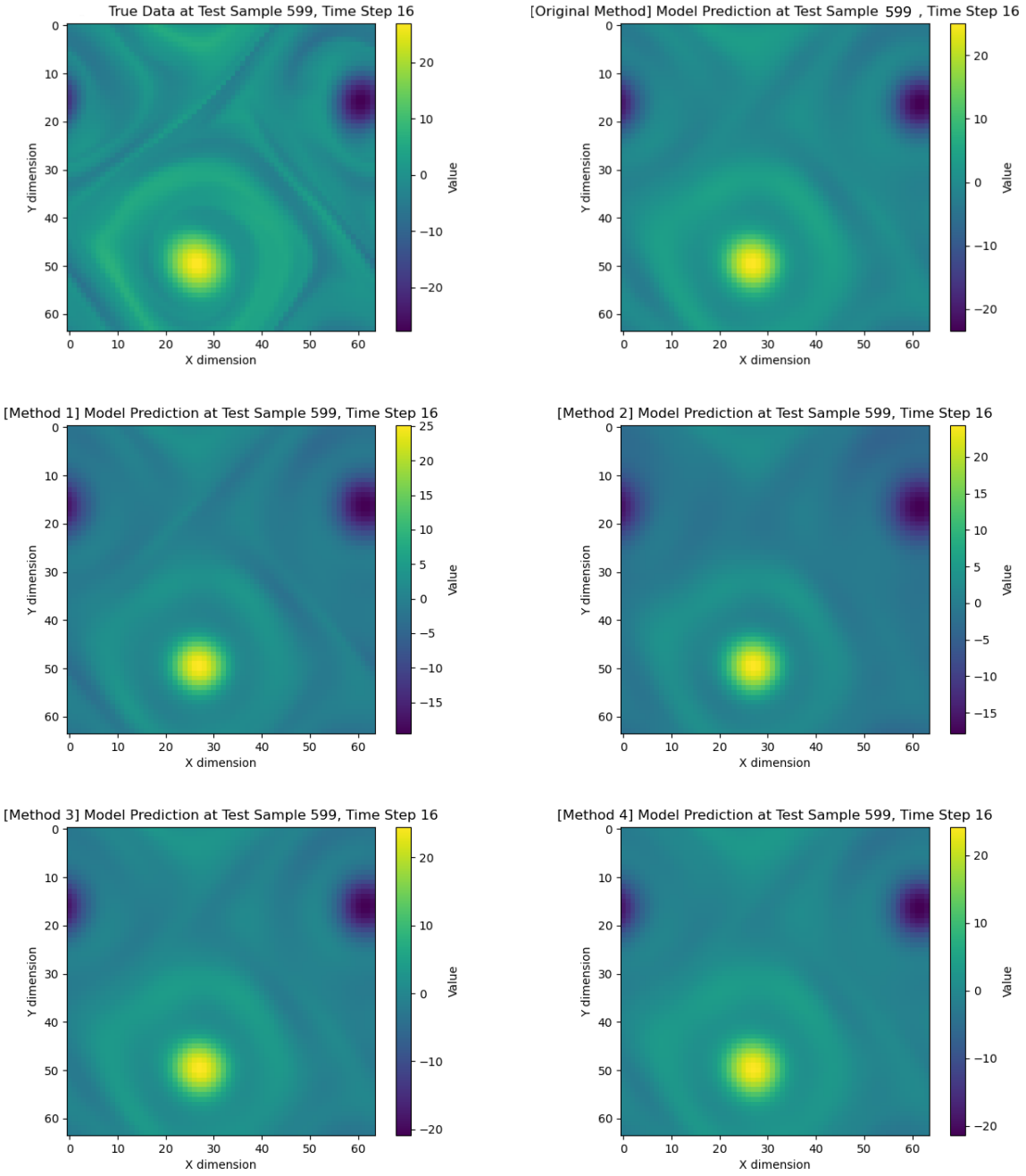

Figure 19: Test Sample 599 at time step 16 (True Data, Original Method, Methods 1-4)

5 Discussion

Though we've justified using training data loss, testing data loss and actual model predictions that Method 1 $\mathcal{L}^s_{\text{data}}$ behaves well in capturing mathematics nature, there are certain concerns.

1. In Time Cost Average for Experiment 3 Fig 15, we actually see an increase in time cost after a similar turning point for all 4 methods, yet the Original Method still decreases. We're yet unable to explain such an 'increase' in time cost, but suggests it has something to do with the 'over-fitting' for gradient matching at the early training stage. To illustrate further on such an issue, more experiments can be conducted via re-designing base learning rates or milestones to reduce learning rates while training.

2. All 4 methods, as shown in weight combinations from Table 2 and Table 3 are chosen separated based on each Norm design. While training the model, we keep them independent of each other, so we have not yet tried to combine different Methods together, for example, use Data Loss function from Method 1 $\mathcal{L}^s_{\text{data}}$ but also put certain weights on PDE Loss function from Method 2 $\mathcal{L}^*_{\text{pde}}$. Further experiments shall be conducted to exhaust as many mixed weight combinations as possible to back up our result.

3. The number of epochs we've run are quite limited due to certain limitations and technical issues, only up to 600. Hence if further experiments are to be conducted, we expect to train with HPC and see results lasting for tens of thousands of epochs and see how well our Method 1 behaves then.

4. To justify our claim in 'Approaching Mathematics Nature in Fluid Mechanics', I believe we need to test on more equations, such as Burger's, Kolmogorov Flows, Euler and etc. Navier-Stokes Equations are indeed a set of PDEs of high difficulty, so one may expect certain norms inherited from regularity theory for these PDEs to work out better and clearer.

6 Conclusion

In this project, we modified the original Data Loss $\mathcal{L}_{\text{data}}$ (2.6) and PDE Loss functions $\mathcal{L}_{\text{pde}}$ (2.5) in PINO that aims to train the FNO Model to learn a PDE operator (2.1), which predicts future behaviors of turbulence given an initial state, as governed by $2D$ Incompressible Navier-Stokes vorticity formulation (2.2). Based on Uniqueness and Existence Theory 2.1, 2.2 in [4] and Sobolev Training in [5], we designed 4 Methods to modify the Loss functions:

- $\mathcal{L}^{s}_{\text{data}}$ Strong Solution Norm (3.2)
- $\mathcal{L}^{*}_{\text{pde}}$ Sobolev PDE Loss (3.3)
- $\mathcal{L}^{w}_{\text{data}}$ Weak Leray-Hopf Solution Norm (3.6)
- $\mathcal{L}^{w,t}_{\text{data}}$ Weak Leray-Hopf Solution Norm with time regularity (3.8)

We tested all 4 methods with a dataset as explained in Section 4.1, and by comparing training data losses, testing data losses, and actual model predictions in Experiment 2 and 3, we've justified that Method 1 $\mathcal{L}^{s}_{\text{data}}$ Strong Solution Norm (3.2) captures the best physics phenomenon governed by Navier-Stokes, even outperforming the Original Method in [3]. Hence we're honored to have introduced a new perspective in designing Data Loss Functions to Neural Operators, which is to encode proper PDE energy class (regularity) choice. This finding leverages the mathematics nature in machine learning to a greater extent, and sheds lights on more future collaborations between mathematical PDE analysts (like me) with data scientists.

References

[1] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Graph kernel network for partial differential equations," 2020.

[2] ——, "Fourier neural operator for parametric partial differential equations," 2021.

[3] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, "Physics-informed neural operator for learning partial differential equations," 2023.

[4] J. Bedrossian and V. Vicol, The mathematical analysis of the incompressible Euler and Navier-Stokes equations : an introduction, ser. Graduate studies in mathematics. American Mathematical Society, 2022. [Online]. Available: https://cir.nii.ac.jp/crid/1130012225924137482

[5] H. Son, J. W. Jang, W. J. Han, and H. J. Hwang, "Sobolev training for physics informed neural networks," 2021.

[6] G. Folland, Real Analysis: Modern Techniques and Their Applications, ser. Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley, 1999. [Online]. Available: https://books.google.com/books?id=N8jVDwAAQBAJ

[7] L. C. Evans, Partial differential equations. Providence, R.I.: American Mathematical Society, 2010.

[8] A. J. Majda and A. L. Bertozzi, Vorticity and Incompressible Flow, ser. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2001.

[9] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, "Sobolev training for neural networks," CoRR, vol. abs/1706.04859, 2017. [Online]. Available: http://arxiv.org/abs/1706.04859

[10] T. Buckmaster and V. Vicol, "Convex integration and phenomenologies in turbulence," 2019.