STRUCTURE PRESERVATION IN NEURAL NETWORKS: APPROXIMATION PROPERTIES AND PARTIAL DIFFERENTIAL EQUATION SOLVERS

GAOHANG CHEN

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University

Department of Applied Mathematics

Structure Preservation in Neural Networks: Approximation Properties and Partial Differential Equation Solvers

Gaohang Chen

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

June, 2025

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Gaohang Chen
_____ (Name of student)

Abstract

This dissertation focuses on incorporating structure-preserving concepts into the design and implementation of neural networks (NNs). As a versatile and quintessential model of deep learning, NNs have been widely applied in various scientific and engineering fields. However, practical applications often rise to the challenge of ensuring the physical or mathematical properties of the underlying problem, leading to external constraints on the network output or training process. Therefore, it is crucial to develop a systematic NN design that can preserve the intrinsic structure while maintaining the flexibility and expressiveness of deep learning.

To address this challenge, we draw inspiration from the structure-preserving concept in numerical analysis, which aims to preserve the intrinsic structures of physical systems, such as conservation laws or symmetries. We incorporate this concept into the design and implementation of NNs, and then investigate the challenges and opportunities that arise from this approach. The structure-preserving properties arose from the practical applications often impose additional challenges, thereby we need to investigate whether NNs can still maintain effectiveness and efficiency under these constraints. Additionally, we explore the implications of exploiting these structure-preserving properties as an inductive bias, which can help to improve the performance and physical fidelity of NN-based models.

Following this discussion, we propose two main research directions in this dissertation. We first consider the approximation properties under a highly constrained training process, which has unique advantages in practical applications. Recent experimental research proposed a novel permutation-based training method, which exhibited a desired classification performance without modifying the exact weight values. In the first part of this dissertation, we provide a theoretical guarantee of this permutation training method by proving its ability to guide a shallow network to approximate any one-dimensional continuous function. Our numerical results further validate this method's efficiency in regression tasks under various initializations. The notable observations during weight permutation suggest that permutation training can provide an innovative tool for describing network learning behavior.

In the second part of this dissertation, we consider the application of solving partial differential equations (PDEs) with NNs, which has shown great potential in various scientific and engineering fields. However, most existing NN solvers mainly focus on satisfying the given PDEs, without explicitly considering intrinsic physical properties, such as mass conservation or energy dissipation. This limitation can result in unstable or nonphysical solutions, particularly in long-term simulations. To address this issue, we propose Sidecar, a novel framework that enhances the physical consistency of existing NN solvers for time-dependent PDEs. Inspired by a traditional structure-preserving numerical approach, our Sidecar framework introduces a small network as a copilot, guiding the primary function-learning NN solver to respect the structure-preserving properties. This framework is designed to be highly flexible, enabling the incorporation of structure-preserving principles from diverse PDEs into a wide range of NN solvers. Our experimental results on benchmark PDEs demonstrate improvements in the accuracy and physical consistency of existing NN solvers.

Publications Arising from the Thesis

[1] Yongqiang Cai, **Gaohang Chen***, & Zhonghua Qiao (2025). Neural networks trained by weight permutation are universal approximators. *Neural Networks*, 187, 107277.

[2] **Gaohang Chen**, Lili Ju, & Zhonghua Qiao* (2025). A Structure-Preserving Framework for Solving Parabolic Partial Differential Equations with Neural Networks. *arXiv preprint* arXiv:2504.10273.

Acknowledgments

I would like to begin by expressing my sincere gratitude to everyone who has supported me throughout this journey. From mentors to friends and family, each of you has played an invaluable role in shaping this experience. Your encouragement, wisdom, and kindness have made this achievement possible, and for that, I am deeply thankful.

First, I would like to express my heartfelt gratitude to my advisor, Professor Zhonghua Qiao, for his invaluable guidance and support in the past three years. His profound expertise, remarkable patience, and personal charm have not only greatly fostered my academic growth but also shaped my character, inspiring me to become a better researcher and person. The topic of this dissertation was based on his insightful suggestions. His encouragement and support generously provided me with numerous opportunities to learn and engage, greatly enriching my research experience. I am grateful for the opportunity to study under his supervision.

I would also like to thank my instructor, Professor Zuowei Shen at the National University of Singapore, for my visit to NUS. From him, I learned the importance of having good taste in research, which has become a guiding principle in my academic journey. His passion for research and unwavering dedication are truly inspiring, and I will always cherish the lessons I have gained under his guidance. I should also thank other faculty members at NUS for their warm welcome and generous support during

my visit, especially Professor Qianxiao Li, whose tremendous assistance greatly contributed to my research.

My sincere appreciation goes to my long-time collaborator and friend, Professor Yongqiang Cai at Beijing Normal University, for his devoted and considerate support for a beginner like me. He has been a fruitful source of inspiration and motivation since I entered the field of machine learning during my master's studies.

I would also like to express my deep gratitude to my late M.S. supervisor, the esteemed Professor Hui Zhang at Beijing Normal University, who introduced me to the world of academic research. Additionally, I am deeply grateful to Professor Zhengru Zhang at Beijing Normal University for her gracious help at the start of my academic journey and for introducing me to my current advisor, Professor Qiao.

During the writing of this dissertation, I have been greatly supported by several experts. I am deeply grateful to Professor Shijun Zhang at HKPolyU for sharing his experience in academic writing, which has been precious in shaping my writing style and approach. My sincere thanks also go to my defense committee members: Professor Shen at NUS, Professor Hongyu Liu at City University of Hong Kong, and Professor Yanping Lin at HKPolyU for their valuable comments and constructive feedback on my dissertation.

I would like to express my deepest gratitude to my family and loved ones for their unconditional support and encouragement in my life and study. Their love and belief in me have been an unwavering source of strength, and I am truly grateful for their presence in my life. I am also deeply grateful to my friends in Hong Kong, Beijing, and Singapore for their warm-hearted help throughout my academic journey. Their companionship and camaraderie have made this journey more enjoyable and fulfilling.

Last but not least, I think I should thank Hong Kong, the lovely city where I have spent the past three years. The vibrant culture, diverse community, and rich history have greatly enriched both my personal life and academic experience. I truly appreciate the opportunity to call this city my home during my Ph.D. journey.

Contents

Abstract v

Publications Arising from the Thesis vii

Acknowledgments ix

1 Introduction 1

1.1 NNs Trained by Weight Permutation are Universal Approximators . . 2

1.2 Sidecar: A Structure-Preserving Framework of Solving PDEs with NNs 3

1.3 Organization of the Dissertation 5

2 Preliminaries 7

2.1 Notations 7

2.2 Neural Networks (NNs) 9

2.3 Approximation Properties of NNs 10

2.4 Training Methods of NNs 12

2.4.1 Traditional Training Methods 13

2.4.2 Permutation Training 14

2.4.3 Advantages of Permutation Training in Hardware Implementation 15

2.5 Solving Partial Differential Equations (PDEs) with NNs 16

2.5.1 Physics-Informed Neural Networks (PINNs) 18

2.5.2 Causal Training Strategy for PINNs 19

2.6 The Structure-Preserving Properties of PDEs 20

2.6.1 Example of Structure-Preserving PDEs 21
2.6.2 The Time-Dependent Spectral Renormalization (TDSR) Method 23

3 UAP via Permutation Training 27
3.1 Introduction 28
3.1.1 Related works 28
3.1.2 Outline 29
3.2 Main Results 29
3.2.1 Architecture of Shallow NNs 29
3.2.2 Weight Configuration and Main Theorems 30
3.2.3 Proof Outline 31
3.3 UAP of Permutation-Trained NNs 34
3.3.1 Approximate the Target Function with a Piecewise Constant Function 34
3.3.2 Construct Step and Constant Function Approximators 35
3.3.3 Annihilate the Unused Part of NNs 39
3.3.4 Proof of Theorem 3.2 42
3.3.5 Proof of Theorem 3.3 45
3.3.6 Estimate the Approximation Rate 48
3.3.7 Proof of Theorem 3.4 50
3.4 Experiments 54
3.4.1 Algorithmic Implementation 54
3.4.2 Experimental Settings 54
3.4.3 Approximating the One-Dimensional Continuous Functions . 56
3.4.4 Approximating the Multi-Dimensional Continuous Functions . 56
3.4.5 Permutation-Trained NNs with Leaky-ReLU 59
3.4.6 The Impact of Permutation Period 60
3.4.7 The Influence of Initialization Strategies 61
3.4.8 Observation of Permutation Patterns 62

4 Sidecar: A Structure-Preserving Framework 65

4.1 Introduction . . . 66

4.1.1 Related Works . . . 66

4.1.2 Outlines . . . 66

4.2 Methodology . . . 67

4.2.1 Framework Architecture . . . 67

4.2.2 Loss Function . . . 68

4.2.3 Training Procedure . . . 71

4.3 Experiments . . . 73

4.3.1 Experimental Setup . . . 73

4.3.2 Dissipative System: Burgers' Equation . . . 74

4.3.3 Conservation System: NLS Equation . . . 76

4.3.4 The Allen-Cahn Equation . . . 81

4.4 Further Discussions . . . 84

4.4.1 The Representation Capacity of Sidecar Architecture . . . 85

4.4.2 The Effectiveness of the Loss Function Design and Implementation . . . 87

4.4.3 The Necessity of the Structure Loss $\mathcal{L}_R$ . . . 88

5 Conclusions and Future Research 91

References 97

List of Tables

Table 3.1: Hyperparameter setting of permutation trained NNs. 55

Table 4.1: The hyperparameter of the Sidecar framework 75

List of Figures

Figure 3.1: Main idea of the construction. (a) Approximate the continuous function f^* by a piecewise constant function g which is further approximated by permuted networks f^{NN}. (b) The step function approximator f_s^{NN} is constructed by step-matching. (c) Refine the basis functions L-times. (d) Stacking pseudo-copies to achieve the desired height. 32

Figure 3.2: Approximating a step function $f_s(x) = 0.8\,\chi(x - 0.4)$ and a constant function $f_c(x) = 0.2$ in $x \in [0, 1]$ with the step-function approximator f_s^{NN} and the constant function approximator f_c^{NN}, respectively. The target functions are plotted as lines, and the approximation results are shown as dashed lines. The locations of the basis functions used for f_s^{NN} and f_c^{NN} are marked with "$\times$" and "$\blacktriangle$" on the x-axis, respectively. 38

Figure 3.3: Approximating one-dimensional continuous function (a): $y = -\sin(2\pi x)$ and (b): $y = \frac{1}{2}(5x^3 - 3x)$ with equidistantly, pairwise random, and randomly initialized network, where $x \in [-1, 1]$. The inset in each panel presents the target function as lines and an example of the approximation result as dots. 57

Figure 3.4: (a) Approximating two-dimensional continuous function $z = -\sin \pi xy$, where $x, y \in [-1, 1] \times [-1, 1]$. The inset panel presents the target function surface and an example of the approximation result as dots. (b) The two-dimensional basis function settings. 58

Figure 3.5: Approximating three-dimensional continuous function $f(x, y, z) = \sin 3x \cdot \cos y \cdot \sin 2z$, where $(x, y, z) \in [-1, 1]^3$. (a) The convergence behavior under random seed 2022. (b) The three-dimensional illustration of the target function, where the function value $f(x, y, z)$ is plotted by the corresponding color in the color bar. 59

Figure 3.6: Approximating one-dimensional continuous function $y = -\sin(2\pi x)$ with equidistantly initialized network equipped with leaky-ReLU, where $x \in [-1, 1]$. The inset in each panel presents the target function as lines and an example of the approximation result as dots. 60

Figure 3.7: Approximating one-dimensional continuous function $y = a_0 + a_1 \sin(\pi x) + a_2 \cos(2\pi x) + a_3 \sin(3\pi x)$ with equidistantly initialized network, where $x \in [-1, 1]$, and the value of permutation period $k = 1, 3, 5, 10, 20$, respectively. The inset in each panel presents the target function as lines and an example of the approximation result as dots. 61

Figure 3.8: The performance of different initialization strategies in approximating $y = -\sin(2\pi x)$ in $[-1, 1]$. The pairwise initialization $\mathbf{w}^{(2n)} = (\pm p_i)_{i=1}^n$, $p_i \sim \mathcal{U}[-1, 1]$ is denoted as $\mathbf{w}^{(2n)} \sim \mathcal{U}^{\pm}[0, 1]^n$. The error bars are omitted for conciseness. The inset panel presents the target function as lines and an example of the approximation result as dots. 62

Figure 3.9: The permutation behavior in the first 400 permutation iterations in approximating $y = -\sin(2\pi x)$ by an equidistantly initialized network with $n = 640$. (a) The distribution of the active components (denoted by dark green color). (b) The frequency distribution illustrates the variation in the total count of active components in each permutation. (c) The corresponding loss behavior. 63

Figure 4.1: The architecture of the Sidecar framework. 68

Figure 4.2: The smooth solution of the Burgers' equation. *Top*: The exact solution of the Burgers' equation. *Bottom*: Comparison of the exact solutions, the vanilla and Sidecar-enhanced PINNs solutions corresponding to the three temporal snapshots. The shown results are the worst cases of the 10 runs. 76

Figure 4.3: The comparison of the vanilla PINNs and the Sidecar-enhanced PINNs for the Burgers' equation. 77

Figure 4.4: The smooth solution of the NLS equation. *Column 1*: The exact solution of the NLS equation. *Columns 2-3*: Comparison of the exact solutions, the vanilla and Sidecar-enhanced PINNs solutions corresponding to the three temporal snapshots. The shown results are the worst cases of the 10 runs. . 78

Figure 4.5: The comparison of the vanilla PINNs and the Sidecar-enhanced PINNs for the NLS equation with mass conservation. 79

Figure 4.6: The comparison of the vanilla PINNs and the Sidecar-enhanced PINNs for the NLS equation with momentum conservation. . . 80

Figure 4.7: The comparison of the momentum/mass conservation performance for the NLS equation between the vanilla PINNs and the Sidecar-enhanced PINNs trained with the mass/conservation law only. 81

Figure 4.8: The solutions of the 1D AC equation (2.14). Left: Illustration of the reference solution. Middle: Comparison of the snapshot at $t = 1$ from the reference solution, the Sidecar-enhanced causal-PINNs, and the equivalent causal-PINNs solutions [86]. The cyan box indicates the region where the solution is zoomed in on the right. Right: The zoomed-in view of the solutions snapshot. The shown results are the worst cases of the 10 runs. 82

Figure 4.9: The comparison of the CausalPINNs and the Sidecar-enhanced CausalPINNs for the Allen-Cahn equation. 84

Figure 4.10: Comparison of copilot-equipped MLPs and the equivalent vanilla MLPs for approximating the reference solutions of the 1D NLS equation Eq. (2.11) and the 1D AC equation Eq. (2.14). 86

Figure 4.11: The comparison of the effectiveness of the Sidecar loss function design. Each point is the result of one random seed. The x- and y-axes represent the MSE error before and after training with the Sidecar loss $\mathcal{L}_{\text{Sidecar}}$, respectively. The point in the lower right corner corresponds to the case where the accuracy of the learned exact solution is further improved by the Sidecar loss $\mathcal{L}_{\text{Sidecar}}$. 88

Figure 4.12: Comparison of the Sidecar-enhanced PINNs with and without the structure loss $\mathcal{L}_R$, and the equivalent vanilla PINNs [72] for 1D NLS equation Eq. (2.11). From left to right: the PINN test loss, the L^2 error of the numerical solution, and the L^∞ error of the mass conservation. 89

Chapter 1

Introduction

As a powerful and versatile tool, neural networks (NNs) have made significant impacts in many fields of scientific and engineering, especially for large-scale and high-dimensional learning problems. Well-designed NN architectures, efficient training algorithms, and advanced hardware implementations have all contributed to the success of deep learning. These advancements have enabled deep learning to achieve state-of-the-art performance in various applications, including image recognition, natural language processing, and generative modeling [33, 54].

However, the black-box nature of NNs makes them difficult to interpret and understand. This lack of controllability can lead to issues in safety, reliability, and trustworthiness, especially in critical applications such as healthcare, finance, and autonomous systems. Additionally, NNs are often used as a surrogate model for complex physical systems, where the input-output relationship is highly constrained by the underlying physics [47]. As a result, there is a growing need for methods that can provide insights into the behavior of NNs and ensure their reliability in real-world applications.

The structure-preserving concepts in numerical analysis provide a reliable perspective for understanding and controlling the behavior of NNs [15, 20, 57, 77]. By incorporating physical constraints and symmetries into the design of NNs, we can create models that are more interpretable, reliable, and efficient. This approach

allows us to leverage the strengths of deep learning while addressing its limitations [37, 61].

This dissertation focuses on incorporating structure-preserving concepts into the design and implementation of NNs. It contains both theoretical and practical aspects and is organized into two main parts. These two parts share a common spirit of enhancing the performance and robustness of NNs by integrating structure-preserving knowledge.

1.1 NNs Trained by Weight Permutation are Universal Approximators

The first part provides a theoretical guarantee of NNs' approximation capability under the external constraint of the training process. Specifically, we consider the Universal Approximation Properties (UAP) of NNs, which serve as a cornerstone in the theoretical guarantee of deep learning. UAP proves that even the simplest one-hidden-layer fully-connected network can approximate any continuous function [18, 39, 56]. This fascinating ability allows NNs to act as surrogate models to replace critical and computationally intensive components in existing methods, thereby improving efficiency [62, 72]. While UAP has been extensively studied in various contexts [10, 23, 85], most existing research assumes that the network parameters can be freely adjusted during the training process. However, specific application scenarios may impose constraints on the network parameters, such as fixed weights or limited weight updates. This constraint is particularly relevant in hardware implementations, where reducing parameter updates can lead to significant power savings [24, 48, 68].

As an extremely constrained scenario, a permutation-based training method has been proposed, where the network weights are initialized randomly and then permuted during training without any updates. Empirical results showed that this

training method can achieve comparable or better performance for image classification tasks [71]. This unique property makes this training method attractive for specific hardware applications, such as fixed-weight accelerators [48] and physical neural networks [68]. While permutation training demonstrates remarkable advantages in practical applications, its lack of theoretical guarantees poses a significant challenge, limiting further advancements in algorithm design and hardware optimization to fully harness its potential.

In the first part of this dissertation, we establish the first theoretical foundation (to our best knowledge) of this permutation training method by proving that a permutation-trained Rectified Linear Unit (ReLU) network with random initializations can achieve the UAP for one-dimensional continuous functions. Compared to the conventional UAP scenarios, the proof of permutation training encounters a significantly non-trivial challenge, primarily due to the extreme constraints of maintaining the initialized weight values. The key idea is a four-pair construction of the step function approximators, which enables the approximation through a piecewise constant function [82]. Additionally, we propose a reorganization method to eliminate the impact of remaining weights.

Our numerical experiments not only validate our theoretical results, which illustrate the widespread existence of the UAP of permutation training in diverse initializations, but also emphasize the importance of initializations on the permutation training performance. Moreover, the patterns observed during permutation training also highlight its potential in describing learning behavior, relating to topics like the pruning technique [26] and continual learning [64, 93].

1.2 Sidecar: A Structure-Preserving Framework of Solving PDEs with NNs

The second part of this dissertation focuses on the application of a structure-preserving framework for solving partial differential equations (PDEs) with NNs.

PDEs are fundamental tools for modeling physical systems, including fluid dynamics, electromagnetism, and quantum mechanics. Since most PDEs do not have analytical solutions, various numerical methods have been developed to obtain approximate solutions with high accuracy and efficiency. In addition to satisfying the PDE formulation, the numerical solution should also consistently respect the intrinsic physical properties of the systems, such as mass conservation and energy dissipation. Therefore, it is essential and highly desirable for numerical solvers to incorporate structure-preserving knowledge into the solving process to ensure the stability and physical fidelity of the solutions, especially for long-term simulations. This principle has been extensively studied in traditional numerical methods, where structure-preserving properties are explicitly embedded into the scheme design, leading to robust and reliable results [15, 20, 57, 77].

With advancements in computational hardware and algorithms, NNs can effectively learn intricate patterns and representations, enabling the development of various NN-based PDE solvers [28, 52, 69, 72, 86, 87, 90, 92]. These solvers use NNs to approximate solution functions directly from the PDE formulation, bypassing the need for high-resolution training data from traditional numerical methods. This makes NN solvers particularly advantageous for applications involving high-dimensional or complex geometries, where conventional numerical approaches often encounter significant difficulties.

However, most existing NN solvers mainly focus on exploiting the given PDE formulation (e.g., minimizing the PDE residual), without explicitly accounting for the intrinsic physical properties of the system. This limitation may result in unstable or nonphysical solutions, reducing the stability and generalization capability of NN solvers. Recent works have attempted to incorporate the structure-preserving knowledge into NN solvers [30, 37, 42, 51], but these approaches often introduce unreasonable trade-offs between accuracy and physical fidelity.

To address this issue, we propose *Sidecar*, a novel framework designed to enhance the physical consistency of existing NN solvers for time-dependent PDEs. The key

idea of Sidecar is to introduce a lightweight NN as a copilot, guiding the primary function-learning NN solver to respect the structure-preserving properties. Both NNs are trained simultaneously, where the primary NN approximates the PDE solution, while the copilot network learns the structure-preserving knowledge via an additional structure loss function. This flexible and plug-and-play design enables Sidecar to cooperate with various NN solvers and adapt to different types of PDEs with diverse structure-preserving properties. The copilot network design is inspired by the Time-Dependent Spectral Renormalization (TDSR) method [17,40], which is a traditional structure-preserving approach for solving time-dependent PDEs. However, implementing TDSR in NN solvers faces notable challenges, particularly in incorporating structure-preserving knowledge and collaborating between the two networks. These challenges are addressed through tailored loss function implementation and training procedures.

In the experiments, we integrate Sidecar with existing NN solvers such as vanilla PINNs [72] and its extensions [86], and then consider benchmark PDEs including conservative systems (nonlinear Schrödinger equation) and dissipative systems (Burgers' equation, Allen-Cahn equation). The resulting solutions show significant enhancement in both accuracy and physical consistency. It demonstrates that Sidecar can effectively incorporate structure knowledge into NN solvers without sacrificing performance. A further discussion provides insights into the advantages of Sidecar, including the architecture design and training procedure, showing its flexibility and robustness in various scenarios. This work also showcases the potential of integrating traditional numerical methods with NN-based techniques to develop more stable and efficient solvers.

1.3 Organization of the Dissertation

The remainder of this dissertation is organized as follows:

- Chapter 2 introduces the notations and essential preliminaries used throughout

the dissertation.

- Chapter 3 shows the UAP of permutation-trained NNs, including proofs and supporting numerical experiments.
- Chapter 4 presents the Sidecar, a structure-preserving framework for NN solvers, detailing its methodology and numerical validation.
- Chapter 5 summarizes the main findings and outlines potential directions for future work.

Chapter 2

Preliminaries

Before moving to the main body of this dissertation, we first introduce the preliminaries, including notations used throughout this dissertation and the necessary background knowledge. We provide a concise overview of neural network (NN) architecture design, their approximation capabilities, their training methods, their applications in solving partial differential equations (PDEs), and the structure-preserving properties inherent to PDEs.

2.1 Notations

We present all notations used throughout this dissertation in this section. Several notations used only in a particular section are not presented here.

- $\mathbb{R}, \mathbb{Z}, \mathbb{N}$ are the set of real numbers, integers, and natural numbers, respectively;
- $\mathbb{R}^+$ is the set of positive real numbers, *i.e.*, $\mathbb{R}^+ = \{x \in \mathbb{R} | x > 0\}$, which also holds for $\mathbb{Z}^+$ and $\mathbb{N}^+$;
- $\{a_i\}_{i=1}^n := \{a_1, \cdots, a_n\}$ is a set with n elements;
- $\{a_i, b_i\}_{i=1}^n := \{a_i\}_{i=1}^n \cup \{b_i\}_{i=1}^n = \{a_1, b_1, a_2, b_2, \cdots, a_n, b_n\}$ is a set with $2n$ elements;

- $\{\pm a_i\}_{i=1}^n := \{+a_1, -a_1, \cdots, +a_n, -a_n\}$ is a set with $2n$ components containing the positive and negative values of a_i;
- $\mathbf{a} = (a_i)_{i=1}^n = (a_1, \cdots, a_n) \in \mathbb{R}^n$ is a vector with n components; It can also been denoted as $\mathbf{a}^{(n)}$ to highlight the length of the vector;
- $(\pm b_i)_{i=1}^n := (+b_1, -b_1, \cdots, +b_n, -b_n) \in \mathbb{R}^{2n}$ is a vector with $2n$ components, where the positive and negative signs are alternated;
- $\mathbf{w}^{(2n)} \sim \mathcal{U}[-1, 1]^{2n}$ means that $\mathbf{w}^{(2n)}$ is a random vector with independent and identically distributed (i.i.d.) components, where each component is uniformly distributed in the interval $[-1, 1]$;
- $\mathbf{W} \in \mathbb{R}^{m \times n}$ is a matrix with m rows and n columns;
- For a function $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{x} \mapsto f(\mathbf{x})$, we use f to denote the function itself, and $f(\mathbf{x})$ to denote the value of the function at $\mathbf{x}$;
- The set of all continuous functions from $\mathbb{X}$ to $\mathbb{Y}$ is denoted by $C(\mathbb{X}, \mathbb{Y})$. When $\mathbb{Y} = \mathbb{R}$, we simply write $C(\mathbb{X})$;
- All the operators and functional are denoted by uppercase calligraphic letters as $\mathcal{A}, \mathcal{B}, \cdots$. The application of an operator $\mathcal{A}$ to a function f is denoted by $\mathcal{A}[f]$, which is also a function. For example, for a function $f : \mathbb{R}^n \to \mathbb{R}$, we denote an operator mapping f to f^2 as $\mathcal{A}$, then

$$\begin{aligned} \mathcal{A}: &\quad (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}), \quad f \mapsto f^2, \\ \mathcal{A}[f]: &\quad \mathbb{R}^n \to \mathbb{R}, \quad \mathbf{x} \mapsto f^2(\mathbf{x}); \end{aligned}$$

- χ is the standard one-dimensional Heaviside function defined as

$$\chi: \quad (\mathbb{R} \to \mathbb{R}), \quad x \mapsto \chi(x) = \begin{cases} 0, & x < 0, \\ 1, & x \geq 0; \end{cases}$$

- For a vector $\mathbf{v}^{(m)} = (v_1, v_2, \cdots, v_m)$, the permuted vector is denoted as $\tau(\mathbf{v}^{(m)}) = (\tau(v_i))_{i=1}^m$, which is a rearrangement of the elements within $\mathbf{v}^{(m)}$;
- $a \sim \mathcal{O}(b)$ means that a is a function of b with the same order, *i.e.*, there exists a constant $C > 0$ such that $|a| \leq C|b|$;
- $a \sim \mathcal{O}(1)$ means that a is at most a constant, *i.e.*, there exists a constant $C > 0$ such that $|a| \leq C$;
- For a d-dimensional PDE, the solution function is denoted as

$$u: \quad \big(\Omega \times [0,T]\big) \to \mathbb{R}, \quad (\mathbf{x}, t) \mapsto u(\mathbf{x}, t),$$

  where $\Omega \subset \mathbb{R}^d$ is the spatial domain and $[0, T]$ is the time domain;
- In the context of PDEs, we denote the partial derivative of u with respect to the spatial variable $\mathbf{x}$ as $u_{\mathbf{x}}$, which is equivalent to $\frac{\partial}{\partial \mathbf{x}} u$. Additionally, the time derivative of u is denoted as u_t, which is equivalent to $\frac{\partial}{\partial t} u$.

2.2 Neural Networks (NNs)

Neural networks (NNs) are powerful and flexible deep learning models that have been widely applied across scientific and engineering domains. There are many different architectures of NNs. In this dissertation, we mainly focus on the multi-layer perceptron (MLP, also called fully-connected NNs, or feedforward NNs) [33], which is a quintessential model in deep learning.

Specifically, the MLP model is an NN that consists of multiple layers of neurons. Here, we denote an MLP as

$$\begin{aligned} f^{\mathrm{NN}}: &\mathbb{R}^d \to \mathbb{R}^n, \\ &\mathbf{x} \mapsto \mathbf{W}_{\mathrm{out}} \sigma(\mathbf{W}_L \sigma(\cdots \sigma(\mathbf{W}_1(\mathbf{x}) + \mathbf{b}_1) \cdots) + \mathbf{b}_L) + \mathbf{b}_{\mathrm{out}}, \end{aligned} \tag{2.1}$$

where the input vector is $\mathbf{x} = (x_1, x_2, \cdots, x_d) \in \mathbb{R}^d$. The number of hidden layers is denoted as L, and the width (*i.e.*, the number of neurons in each hidden layer) is denoted as W_i, $i = 1, 2, \cdots, L$. The weights and biases of the i-th hidden layer are denoted as $\mathbf{W}_i \in \mathbb{R}^{W_{i+1} \times W_i}$ and $\mathbf{b}_i \in \mathbb{R}^{W_{i+1}}$, respectively. The widths of the hidden layers are often chosen to be the same, *i.e.*, $W_i = W$, $\forall i = 1, 2, \cdots, L$. Additionally, the output layer is denoted as $\mathbf{W}_{\text{out}} \in \mathbb{R}^{n \times W_L}$ and $\mathbf{b}_{\text{out}} \in \mathbb{R}^n$ to match the output dimension. The trainable parameters of the MLP are the weights and biases, which are denoted as $\boldsymbol{\theta} = \{\mathbf{W}_i, \mathbf{b}_i\}_{i=1}^L \cup \{\mathbf{W}_{\text{out}}, \mathbf{b}_{\text{out}}\}$.

Here σ is the activation function. Common choices for the activation function include

- Sigmoid function: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$;
- Hyperbolic tangent function: $\sigma(z) = \tanh(z) = \dfrac{e^x - e^{-x}}{2}$;
- Rectified linear unit (ReLU) function: $\sigma(z) = \text{ReLU}(z) = \max\{0, z\}$.

Notice that ReLU activation is positively homogeneous *i.e.*, $\text{ReLU}(\lambda\, x) = \lambda\, \text{ReLU}(x)$ for all $\lambda > 0$. Therefore, in the special case of one-dimensional functions and one-hidden-layer MLP, where the MLP has the form of a linear combination of ReLU basis functions like

$$f(x) = \sum_{i=1}^{W} a_i \text{ReLU}(w_i x + b_i) + c,$$

and then the weights w_i can be chosen as $w_i = \pm 1$ without loss of generality.

For a multi-dimensional vector $\mathbf{z} = (z_1, z_2, \cdots, z_n)$, the activation function is applied component-wise, *i.e.*, $\sigma(\mathbf{z}) = \big(\sigma(z_1), \sigma(z_2), \cdots, \sigma(z_n)\big)$.

2.3 Approximation Properties of NNs

As a cornerstone in the theoretical guarantee of deep learning, the universal approximation property (UAP) of NNs states that a sufficiently large NN can approximate any continuous function to any desired accuracy [18, 39, 56]. It evaluates

the ability of NNs to represent functions within a target function space. If the *hypothesis space*, which is defined as the set of functions that can be represented by an NN with a given architecture, is dense in the target function space, then the NN is said to have the UAP. It is important to note that the UAP of NNs does not depend on the specific training algorithms or data sampling strategies; it is determined solely by the network architecture and the choice of target function space. Therefore, it serves as both a fundamental and ideal scenario for evaluating the performance of NNs.

The UAP of NNs has been extensively studied in the literature. The first foundational result of UAP was established for MLPs with a single hidden layer and arbitrary width [18, 39, 56]. It can be stated as

Theorem 2.1 (UAP of MLPs with a single hidden layer, [56]). *For any $m, n \in \mathbb{N}$, compact set $K \subset \mathbb{R}^d$, continuous function $f \in C(\Omega, \mathbb{R}^m)$, non-polynomial activation functions $\sigma \in C(\mathbb{R}, \mathbb{R})$, and $\varepsilon > 0$, there exists $k \in \mathbb{N}$, $\mathbf{W} \in \mathbb{R}^{k\times n}$, $\mathbf{b} \in \mathbb{R}^k$, and $\mathbf{A} \in \mathbb{R}^{m\times k}$, such that*

$$\sup_{\mathbf{x}\in K} \|f(\mathbf{x}) - \mathbf{A} \cdot \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})\| < \varepsilon.$$

This theorem can be proved through several methods, such as interpreting the single-hidden-layer MLP as a linear combination of basis functions or applying the Stone-Weierstrass theorem [82]. However, this result does not impose constraints on the parameters of the MLP, as it abstracts away from the specific training process. While it guarantees the existence of a solution for the approximation problem, it does not provide a constructive method for designing NNs that achieve the UAP.

Further research primarily concentrated on how the width and depth of NNs influence their UAP. The optimal approximation rate of MLPs equipped with ReLU activation functions has been established in terms of the width and depth, which is given by the following theorem in [80]:

Theorem 2.2 (Optimal approximation rate of MLPs with ReLU activation, [80]).

Given a continuous function $f \in C([0,1]^d)$*, for any* $N \in \mathbb{N}^+$*,* $L \in \mathbb{N}^+$*, and* $p \in [1,\infty]$*, there exists a function* ϕ *implemented by a ReLU network with width* $C_1 \max\left\{d\lfloor N^{1/d}\rfloor, N+2\right\}$ *and depth* $11L + C_2$*, such that*

$$\|f - \phi\|_{L_p([0,1]^d)} \leq 131\sqrt{d}\,\omega_f\left(\left(N^2L^2\log_3(N+2)\right)^{-1/d}\right),$$

where $C_1 = 16$ *and* $C_2 = 18$ *if* $p \in [1,\infty)$*;* $C_1 = 3^{d+3}$ *and* $C_2 = 18 + 2d$ *if* $p = \infty$*. The function* ω_f *is the modulus of continuity of* f *defined as*

$$\omega_f(\delta) = \sup\{|f(\mathbf{x}) - f(\mathbf{y})| : \mathbf{x},\mathbf{y} \in [0,1]^d, \|\mathbf{x}-\mathbf{y}\| \leq \delta\}.$$

Other related works include estimating the lower bound for the minimum width required by NNs to achieve UAP [5,63]. Additionally, MLPs with specific activation functions can achieve UAP with fixed width and depth [95].

The following works also studied the UAP of different target function spaces, such as the UAP of continuous functional or operator [13], dynamical systems [21], and functions with certain symmetry [59]. Additionally, the UAP of NNs has been extended to other architectures, including convolutional neural networks (CNNs) [14,35] and residual networks (ResNets) [58].

2.4 Training Methods of NNs

Training a NN is to find the optimal (or good enough) parameters $\boldsymbol{\theta}$ that maximize the NN's prediction accuracy. It can be formulated as an optimization problem, where the goal is to minimize the loss function $\mathcal{L}(\boldsymbol{\theta})$. The loss function $\mathcal{L}$ evaluates the discrepancy between the predicted output and the true output. The specific form of $\mathcal{L}$ depends on the nature of the problem being addressed. For example, in a regression problem, the loss function is often defined as the mean squared error

(MSE) between the predicted output f^{NN} and the true output f, given by

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(f^{\text{NN}}(\mathbf{x}_i, t_i) - f(\mathbf{x}_i, t_i) \right)^2, \tag{2.2}$$

where N is the number of training samples, $\{(\mathbf{x}_i, t_i), f(\mathbf{x}_i, t_i)\}_{i=1}^{N}$ are the training samples, where $f(\mathbf{x}_i, t_i)$ are the true output corresponding to the input $(\mathbf{x}_i, t_i)$.

2.4.1 Traditional Training Methods

The traditional training process is typically performed using a *gradient descent algorithm*, which updates the parameters $\boldsymbol{\theta}$ within the NN in the direction of the negative gradient of the loss function. The update rule for the weights and biases is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, \mathcal{G}_{\boldsymbol{\theta}}, \tag{2.3}$$

where η is the learning rate, which can be adjusted during the training process to improve convergence. Here $\mathcal{G}_{\boldsymbol{\theta}}$ is the gradient computed using back-propagation [74]. If the gradients are computed using the entire training set, the update is called *full-batch* gradient descent, which is often computationally expensive. In practice, a *mini-batch* gradient descent is often used, where the gradients are computed using a small subset of the training data (called a mini-batch) instead of the entire dataset. This approach reduces the computational cost and can lead to faster convergence. Further improvements can be made by using adaptive learning rate methods such as Adam [46], which adjust the learning rate based on the historical gradients.

The update in Eq. (2.3) is repeated for several epochs until the performance of NNs reaches a satisfactory level, or the maximum number of epochs is reached. The pseudocode of the Adam-based traditional training process is shown in Algorithm 1.

Algorithm 1: Adam-based traditional training algorithm

Input: Loss function $\mathcal{L}$, training set D_T, maximum training epoch M, optimizer Adam, learning rate η, initial weights $\mathbf{w}$

Output: Trained weights $\boldsymbol{\theta}_M$

$\boldsymbol{\theta}_0 \leftarrow \mathbf{w}$; `// Initialize the weights`

`// Training iteration`

for $t = 1, 2, \ldots, M$ **do**

  $\boldsymbol{\theta}_t \leftarrow \text{Adam}(\mathcal{L}, \boldsymbol{\theta}_{t-1}, D_T)$; `// Update the weights by Adam`

return $\boldsymbol{\theta}_M$;

2.4.2 Permutation Training

The traditional training methods discussed above have no constraint on the parameters within NNs, leading to the scenario called *free training*. However, in some applications, especially in specific hardware implementations, the parameters of NNs are preferably constrained to a certain range or structure.

As an extreme constrained scenario, NNs are trained by rearranging the initialized parameters without altering their exact values [71]. This can be described as a permutation of the initialized weights, thereby we call it *permutation training*.

Definition 2.3. For a vector $\mathbf{v}^{(m)} = (v_1, v_2, \cdots, v_m)$, the permutation τ is a bijection from the element set $\{v_i\}_{i=1}^m$ to itself.

In the implementation of permutation training, guidance is crucial in finding the ideal order relationship of weights. The algorithm *lookahead permutation (LaPerm)* [71] introduces an k-times Adam-based free updating. This algorithm rearranges the initial weights $\boldsymbol{\theta}_0$ by the order relationship learned during the free training process. Therefore, the trained weights $\boldsymbol{\theta}_T$ can be viewed as a permutation of the initial weights $\boldsymbol{\theta}_0$. The pseudocode is shown in Algorithm 2.

Permutation training has a unique property that distinguishes it from traditional training methods. Notice that all permutations of $\mathbf{v}^{(m)}$ can form a group denoted

Algorithm 2: Adam-based LaPerm algorithm

Input: Loss function $\mathcal{L}$, training set D_T, maximum training epoch M, Inner optimizer Adam, permutation period k, initial weights $\mathbf{w}$

Output: Trained weights $\boldsymbol{\theta}_M$

$\boldsymbol{\theta}_0 \leftarrow \mathbf{w}$; `// Initialize the weights`

`// Training iteration`

for $t = 1, 2, \ldots, M$ **do**

  $\boldsymbol{\theta}_t \leftarrow \mathrm{Adam}(\mathcal{L}, \boldsymbol{\theta}_{t-1}, D_T)$; `// Free training by Adam`

  if k *divides* t **then**

    $\boldsymbol{\theta}_t \leftarrow \tau_t(\mathbf{w})$; `// Apply the permutation`

return $\boldsymbol{\theta}_M$;

as S_m, which is closed under composition. Precisely, for any $\pi, \tau \in S_m$, there is a $\rho \in S_m$ such that $\rho(v_i) = \pi(\tau(v_i))$ for all v_i within $\mathbf{v}^{(m)}$. This property leads to the major advance of permutation training: ideally, the final weight $\boldsymbol{\theta}_T$ can be achieved by permuting the initialized weight $\boldsymbol{\theta}_0$ only once, *i.e.*, there exists a $\tau \in S_m$ such that $\boldsymbol{\theta}_T = \tau(\boldsymbol{\theta}_0)$. This *one-step* nature significantly distinguishes permutation training from other iterative training methods like Adam.

2.4.3 Advantages of Permutation Training in Hardware Implementation

Permutation training in Algorithm 2 has been applied to image classification tasks [71], achieving comparable or even superior performance to conventional free-training training methods like Adam. However, there is currently no evidence to report a significant advantage when applied to more diverse tasks on contemporary Graphics Processing Unit (GPU)-based hardware. Nevertheless, we believe it is highly suitable for the design of *physical neural networks* [68], as reducing the number of weight updates can significantly lower the power consumption during the

training process. Therefore, permutation training may inspire alternative physical weight connection implementations, such as using fixed-weight devices controlled by a permutation circuit [71]. This idea has been applied to a fixed-weight network accelerator [48, 49].

Another potential application scenario is the hardware with an explicit structure to store the weight value, such as the *integrated photonic tensor core* [24]. In this novel computing chip, an array of phase-change cells to separately store each element of the weight matrix, with their values adjusted through optical modulation of the transmission states of the cells. This design has been successfully employed by international commercial companies in their photonic computing products. However, permutation training indicates that, in addition to changing the exact value, it is feasible to connect each cell with the permutation circuit for convenient reconnections. This flexibility can significantly enhance the learning process by leveraging permutation training.

2.5 Solving Partial Differential Equations (PDEs) with NNs

As a typical scientific and engineering application, developing efficient and accurate NN-based PDE solvers has attracted increasing interest in recent years. PDEs serve as essential mathematical models for describing physical systems. Since most PDEs do not have closed-form exact solutions, numerical methods have been developed to compute approximate solutions with high accuracy and efficiency. For a general time-dependent PDE of $u(\mathbf{x}, t)$ in the form

$$\begin{cases} u_t = \mathcal{A}[u], & (\mathbf{x}, t) \in \Omega \times [0, T], \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}, t) = g(\mathbf{x}, t), & (\mathbf{x}, t) \in \partial\Omega \times [0, T]. \end{cases} \tag{2.4}$$

Here $\mathbf{x} = (x_1, x_2, \cdots, x_d) \in \Omega \subset \mathbb{R}^d$ is the spatial coordinate, $t \in [0, T]$ is the temporal coordinate, $\mathcal{A}$ is a known operator, $\Omega \in \mathbb{R}^d$ is the spatial domain, T is the final time, $u_0(\mathbf{x})$ is the given initial condition defined on the spatial domain Ω, and $g(\mathbf{x}, t)$ is the given boundary condition defined on the boundary $\partial\Omega \times [0, T]$ of the spatiotemporal domain.

For clarity, we denote u as the exact solution of the PDE and $\bar{u}$ as the approximate solution obtained by the NN solvers. The goal is to find the solution function $\bar{u}(\mathbf{x}, t)$ that satisfies Eq. (2.4). This task can be viewed as a direct application of the UAP of NNs to different target function spaces, and can be categorized into two main approaches: *function learning* and *operator learning*.

Function Learning Methods

Thanks to the function-UAP of NNs [18,39,56], the function learning approaches [72,90,92] focus on approximating the solution function $u(\mathbf{x}, t)$ directly using NNs, which is based on the function-UAP of NNs. A key advantage of these function approximation-based methods is their ability to learn solutions directly from the given PDE formulation, eliminating the need for high-resolution training data from traditional solvers. These methods are particularly advantageous for practical applications involving high-dimensional or complex geometries, where traditional numerical approaches often face challenges.

Operator Learning Methods

NNs have also been proven to have UAP for operators [13], which inspires the development of operator learning methods for PDEs [50,60,62]. These methods can neither learn the semi-discretized evolution operators [60], nor learn the operators from the given conditions (such as initial conditions, boundary conditions, or coefficients) to the solutions [62]. In this way, the learned operator can handle different given conditions or coefficients without requiring retraining. This approach also shows potential in solving inverse problems, *i.e.*, inferring conditions or coefficients

from the given solutions [34, 66]. However, learning operators is significantly more challenging than learning solution functions, as it often demands a large amount of training data.

2.5.1 Physics-Informed Neural Networks (PINNs)

The most commonly used function learning method is PINNs [45, 72] and its extensions [28, 52, 69, 86, 87]. The vanilla PINNs [72] adopt a MLP in Eq. (2.1) to approximate the solution function $u(\mathbf{x}, t)$, where the input vector is aligned as $(\mathbf{x}, t) = (x_1, x_2, \cdots, x_d, t)$. To ensure the differentiability of the solution function, the activation function σ is often chosen as the hyperbolic tangent function $\sigma(\mathbf{x}) = \tanh(\mathbf{x})$. The loss function is designed to minimize the mean square L^2-norm (also called MSE) of the PDE residual, *i.e.*, the difference between the left-hand side and the right-hand side of the PDEs. The initial and boundary conditions are also incorporated into the loss function to ensure that the solutions satisfy the given conditions. For the general PDE system in Eq. (2.4), the loss function of PINNs can be written as

$$\mathcal{L}_{\text{PINNs}}[\bar{u}] = \mathcal{L}_{\text{PDE}}[\bar{u}] + \mathcal{L}_{\text{data}}[\bar{u}], \tag{2.5}$$

where

$$\begin{aligned} \mathcal{L}_{\text{PDE}}[\bar{u}] &= \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left| \bar{u}_t(\mathbf{x}_i, t_i) - \mathcal{A}[\bar{u}](\mathbf{x}_i, t_i) \right|^2, \\ \mathcal{L}_{\text{data}}[\bar{u}] &= \frac{1}{N_{\text{IC}}} \sum_{j=1}^{N_{\text{IC}}} \left| \bar{u}(\mathbf{x}_j, 0) - u_0(\mathbf{x}_j) \right|^2 + \frac{1}{N_{\text{BC}}} \sum_{k=1}^{N_{\text{BC}}} \left| \bar{u}(\mathbf{x}_k, t_k) - g(\mathbf{x}_k, t_k) \right|^2. \end{aligned} \tag{2.6}$$

Here $\{(\mathbf{x}_i, t_i)\}_{i=1}^{N_{\text{PDE}}} \in \Omega \times [0, T]$ are the collocation points for the PDE residual loss, and $\{(\mathbf{x}_j, 0)\}_{j=1}^{N_{\text{IC}}} \in \Omega$ and $\{(\mathbf{x}_k, t_k)\}_{k=1}^{N_{\text{BC}}} \in \partial\Omega \times [0, T]$ are for the initial and boundary conditions, respectively. These collocation points can be randomly sampled, making PINNs a mesh-free method. Notice that the loss function Eq. (2.5) depends solely

on the given PDE formulation and its associated conditions, without requiring explicit or high-resolution numerical solutions. This characteristic greatly enhances the practical applicability of PINNs.

2.5.2 Causal Training Strategy for PINNs

There are some advanced techniques to improve the performance and training efficiency of PINNs, such as adaptive sampling strategy [28, 52] and learning rate annealing algorithm [87]. One insightful technique is the causal training strategy [86], which encourages PINNs to learn the solution in accordance with the temporal causality of the PDEs. To illustrate this idea, we discretize the time domain $[0, T]$ into N_T time points as $\{t^n\}_{n=0}^{N_T}$, and define the residual loss at each time point t^n as

$$\mathcal{L}_{\mathrm{PDE}}^n[\bar{u}] = \frac{1}{N_n} \sum_{i=1}^{N_n} \left| \bar{u}_t(\mathbf{x}_i, t^n) - \mathcal{A}[\bar{u}](\mathbf{x}_i, t^n) \right|^2,$$

where N_n is the number of spatial collocation points at time t^n. Under this setting, the overall PDE residual loss can be written as $\mathcal{L}_{\mathrm{PDE}}[\bar{u}] = \frac{1}{N_T} \sum_{n=0}^{N_T} \mathcal{L}_{\mathrm{PDE}}^n[\bar{u}]$. However, to respect the temporal causality, the residual loss is reformulated as a weighted form:

$$\widetilde{\mathcal{L}}_{\mathrm{PDE}}[\bar{u}] = \frac{1}{N_T} \sum_{n=0}^{N_T} w_n \, \mathcal{L}_{\mathrm{PDE}}^n[\bar{u}], \text{ where } w_n = \exp\left(-\varepsilon \sum_{l=0}^{n-1} \mathcal{L}_{\mathrm{PDE}}^l[\bar{u}] \right). \tag{2.7}$$

Here the temporal weights w_n are designed to be small unless all the previous time points $\{t^l\}_{0 \le l < n}$ are well-approximated, and ε is a hyperparameter that controls the decay rate of the weights (*i.e.*, the larger ε indicates the higher accuracy requirement for the previous time points). The causal training strategy can be easily integrated into the existing PINNs solvers, and has shown great potential in improving the performance of the PINNs for PDEs with strong temporal dependencies.

2.6 The Structure-Preserving Properties of PDEs

The structure-preserving properties of PDEs are the intrinsic physical laws that the solutions satisfy. For a general temporal evolution PDE of $u(\mathbf{x}, t)$ in a form given by Eq. (2.4), which is restated as

$$\begin{cases} u_t = \mathcal{A}[u], & (\mathbf{x}, t) \in \Omega \times [0, T], \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}, t) = g(\mathbf{x}, t), & (\mathbf{x}, t) \in \partial\Omega \times [0, T]. \end{cases}$$

The systems may have some intrinsic physical properties, which are often described by the evolution of preserved quantities over time with a certain speed:

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\mathcal{Q}[u] = \mathcal{S}[u], \\ \mathcal{Q}[u](0) = \mathcal{Q} \circ \iota[u_0] =: C_0, \end{cases} \tag{2.8}$$

where $\mathcal{Q}, \mathcal{S} : (\Omega \times [0, T] \to \mathbb{R}) \to ([0, T] \to \mathbb{R})$ is the quantity of interest and its evolution speed, respectively. The initial value C_0 is determined by the initial condition $u_0(\mathbf{x})$, where ι is the natural embedding operator

$$\iota : (\Omega \to \mathbb{R}) \to (\Omega \times [0, T] \to \mathbb{R}), \quad u_0(\cdot) \mapsto u(\cdot, 0),$$

and $\circ$ denotes the composition of operators. Eq. (2.8) holds for both conservative and dissipative systems:

- **Conservative systems:** The preserved quantities remain unchanged over time, *i.e.*, $\mathcal{S}[u] = 0$ and $\mathcal{Q}[u](t) \equiv C_0$, $\forall t \in [0, T]$;
- **Dissipative systems:** The preserved quantities decay over time at a rate given by $\mathcal{S}[u] < 0$.

Our goal is to find an approximate solution $\bar{u}(\mathbf{x}, t)$ that satisfies the PDE Eq. (2.4) as well as the structure-preserving properties Eq. (2.8), ensuring that the solutions

are stable, accurate, and physically meaningful.

2.6.1 Example of Structure-Preserving PDEs

Example 2.1 (Burgers' equation)**.** Burgers' equation is a well-known nonlinear PDE in various physical phenomena such as fluid dynamics and traffic flow [88]. The one-dimensional Burgers' equation of $u(x,t)$ as

$$\begin{cases} u_t + uu_x - \nu u_{xx} = 0, & (x,t) \in [-1,1] \times [0,1], \\ u(x,0) = u_0(x), & x \in [-1,1], \\ u(-1,t) = u(1,t) = 0, & t \in [0,1], \end{cases} \tag{2.9}$$

where ν is the viscosity coefficient. The case of $\nu = 0$ corresponds to the inviscid Burgers' equation, which is a conservative system. Burgers' equation with $\nu > 0$ is a classical example of a dissipative system, in which the total energy decays over time. By multiplying Eq. (2.9) by u and integrating over the domain $[-1,1]$, we obtain the following dissipation law of the energy $\mathcal{E}_B[u]$:

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\mathcal{E}_B[u] = \mathcal{S}_B[u], \\ \mathcal{E}_B[u](0) = C_0, \end{cases} \quad \text{where} \quad \begin{aligned} \mathcal{E}_B[u](t) &:= \int_{-1}^{1} u^2(x,t)\,\mathrm{d}x, \\ \mathcal{S}_B[u](t) &:= -2\nu \int_{-1}^{1} u_x^2(x,t)\,\mathrm{d}x, \end{aligned} \tag{2.10}$$

and $C_0 = \mathcal{E}_B \circ \iota[u_0] = \int_{-1}^{1} u_0^2\,\mathrm{d}x$. This equation describes the evolution of the total energy $\mathcal{E}_B[u]$ of the system, which decays over time with the speed $\mathcal{S}_B[u]$. Ideally, the approximate solution $\bar{u}(x,t)$ should satisfy both the Burgers' equation (2.9) and the energy dissipation rates Eq. (2.10).

Example 2.2 (Nonlinear Schrödinger equationl, NLS)**.** The NLS equation is a complex-valued PDE system with the form in one dimension as

$$iu_t + \frac{1}{2}u_{xx} = \kappa|u|^2 u, \quad (x,t) \in [-15,15] \times [0,\pi/2], \tag{2.11}$$

where $u(x,t) \in \mathbb{C}$ is the complex-valued wave function, and $|\cdot|$ denotes the norm of the complex number. We choose $\kappa = -1$ to ensure the stability of the solution [88], along with the periodic boundary conditions as

$$\begin{aligned} u(x,0) &= u_0(x), \\ u(-15,t) &= u(15,t), \\ u_x(-15,t) &= u_x(15,t). \end{aligned}$$

The NLS equation has several preserved quantities, one of which is the mass conservation law [19], *i.e.*, the total probability density of the wave function remains constant over time, which is given by

$$\mathcal{Q}_1[u](t) := \int_{-15}^{15} |u(x,t)|^2 \, \mathrm{d}x = C_0^{(1)}, \quad \text{where } C_0^{(1)} = \int_{-15}^{15} |u_0(x)|^2 \, \mathrm{d}x. \tag{2.12}$$

Another important conserved quantity is the total momentum of the wave function, which is defined as

$$\mathcal{Q}_2[u](t) := \mathrm{Im}\left[\int_{-15}^{15} u^*(x,t)\, u_x(x,t)\, \mathrm{d}x\right] = \int_{-15}^{15} \big(\mathrm{Re}[u]\cdot \mathrm{Im}[u_x] - \mathrm{Im}[u]\cdot \mathrm{Re}[u_x]\big)\, \mathrm{d}x,$$

where $u^* = \mathrm{Re}[u] - i \cdot \mathrm{Im}[u]$ is the complex conjugate of u. The corresponding momentum conservation law gives

$$\mathcal{Q}_2[u](t) = C_0^{(2)}, \quad \text{where} C_0^{(2)} = \int_{-15}^{15} \mathrm{Re}[u_0]\cdot \mathrm{Im}[(u_0)_x] - \mathrm{Im}[u_0]\cdot \mathrm{Re}[(u_0)_x]\, \mathrm{d}x. \tag{2.13}$$

Example 2.3 (Allen-Cahn equation)**.** The Allen-Cahn equation is a typical phase-field model for the phase transition phenomena [2]. The one-dimensional form is given as

$$\begin{cases} u_t = \varepsilon^2 u_{xx} + f[u], & (x,t) \in [-1,1] \times [0,1], \\ u(x,0) = u_0(x), & \\ u(-1,t) = u(1,t), \quad u_x(-1,t) = u_x(1,t), & \end{cases} \tag{2.14}$$

where ε reflects the width of the transition regions, and $f[u]$ is a reaction source. As a typical gradient flow model, the Allen-Cahn equation satisfies the energy dissipation law. Specifically, the energy functional is defined as

$$\mathcal{E}_{AC}[u](t) := \int_{-1}^{1} \left(\frac{\varepsilon^2}{2} |u_x(x,t)|^2 + F[u](x,t) \right) \mathrm{d}x, \tag{2.15}$$

where $F[u]$ is the double-well potential function (*i.e.*, $-F' = f$). Therefore, the solution to Eq. (2.14) should decrease the energy Eq. (2.15) over time, *i.e.*,

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\mathcal{E}_{AC}[u] = \mathcal{S}_{AC}[u], \\ \mathcal{E}_{AC}[u](0) = C_0, \end{cases} \tag{2.16}$$

where

$$\mathcal{S}_{AC}[u] := -\int_{-1}^{1} u_t^2(x,t)\,\mathrm{d}x \leq 0$$

and $C_0 = \mathcal{E}_{AC} \circ \iota[u_0] = \int_{-1}^{1} \left(\frac{\varepsilon^2}{2}(u_0)_x^2 + F[u_0] \right) \mathrm{d}x$.

2.6.2 The Time-Dependent Spectral Renormalization (TDSR) Method

The TDSR method [17,40] is a structure-preserving technique for traditional numerical scheme, which introduce a time-dependent factor to incorporate the structure equation Eq. (2.8) into the PDE Eq. (2.4).

The idea of TDSR starts with the observation that the preserved quantities $\mathcal{Q}[u]$ and its evolution speed $\mathcal{S}[u]$ in Eq. (2.8) are often global, *i.e.*, with integration over the spatial domain Ω (such as the energy dissipation law Eq. (2.10) of the Burgers' equation). Therefore, we can assume both $\mathcal{Q}[u]$ and $\mathcal{S}[u]$ have the integration form as

$$\mathcal{Q}[u](t) = \int_{\Omega} \mathcal{K}_{\mathcal{Q}}[u](\mathbf{x},t)\,\mathrm{d}\mathbf{x}, \quad \mathcal{S}[u](t) = \int_{\Omega} \mathcal{K}_{\mathcal{S}}[u](\mathbf{x},t)\,\mathrm{d}\mathbf{x},$$

where $\mathcal{K}_{\mathcal{Q}}$, $\mathcal{K}_{\mathcal{S}} : (\Omega \times [0,T] \to \mathbb{R}) \to (\Omega \times [0,T] \to \mathbb{R})$ are known operators serve as

the integration kernel of $\mathcal{Q}$ and $\mathcal{S}$, respectively. The structure equation (2.8) is then transformed into the integration form as

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\displaystyle\int_\Omega \mathcal{K}_\mathcal{Q}[u](\mathbf{x},t)\,\mathrm{d}\mathbf{x} = \int_\Omega \mathcal{K}_\mathcal{S}[u](\mathbf{x},t)\,\mathrm{d}\mathbf{x}, \\ \displaystyle\int_\Omega \mathcal{K}_\mathcal{Q}[u](\mathbf{x},0)\,\mathrm{d}\mathbf{x} = C_0, \end{cases} \tag{2.17}$$

where $C_0 = \int_\Omega \mathcal{K}_\mathcal{Q}\circ\iota[u_0](\mathbf{x})\,\mathrm{d}\mathbf{x}$. Notice that after integrating over the spatial domain Ω, the structure equation (2.17) only depends on the temporal variable t. Therefore, we can introduce a time-dependent factor $R(t)$ by applying a variable transformation

$$u(\mathbf{x},t) = R(t)\,v(\mathbf{x},t),$$

such that the structure equation (2.8) can be merged into the PDE Eq. (2.4) as a coupled system:

$$\begin{cases} \dfrac{\partial}{\partial t}(R\,v) = \mathcal{A}[R\,v], \\ \dfrac{\mathrm{d}}{\mathrm{d}t}\displaystyle\int_\Omega \mathcal{K}_\mathcal{Q}[R\,v](\mathbf{x},t)\,\mathrm{d}\mathbf{x} = \int_\Omega \mathcal{K}_\mathcal{S}[R\,v](\mathbf{x},t)\,\mathrm{d}\mathbf{x}. \end{cases} \tag{2.18}$$

Since $R(t)$ can be treated as a constant within the spatial integration, we factor out $R(t)$ from the integration kernel $\mathcal{K}_\mathcal{Q}$ and $\mathcal{K}_\mathcal{S}$ as

$$\begin{aligned} \int_\Omega \mathcal{K}_\mathcal{Q}[R\,v](\mathbf{x},t)\,\mathrm{d}\mathbf{x} &= \mathcal{F}_\mathcal{Q}[R](t)\int_\Omega \mathcal{K}^v_\mathcal{Q}[v](\mathbf{x},t)\,\mathrm{d}\mathbf{x}, \\ \int_\Omega \mathcal{K}_\mathcal{S}[R\,v](\mathbf{x},t)\,\mathrm{d}\mathbf{x} &= \mathcal{F}_\mathcal{S}[R](t)\int_\Omega \mathcal{K}^v_\mathcal{S}[v](\mathbf{x},t)\,\mathrm{d}\mathbf{x}, \end{aligned}$$

where $\mathcal{F}_\mathcal{Q}$, $\mathcal{F}_\mathcal{S} : ([0,T]\to\mathbb{R}) \to ([0,T]\to\mathbb{R})$ are the factors depending on $R(t)$, and $\mathcal{K}^v_\mathcal{Q}$, $\mathcal{K}^v_\mathcal{S} : (\Omega\times[0,T]\to\mathbb{R}) \to (\Omega\times[0,T]\to\mathbb{R})$ are the renormalized integration kernels depending on $v(\mathbf{x},t)$. Therefore, the structure equation can be rewritten

into an ordinary differential equation (ODE) for $R(t)$ as

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\big(\mathcal{F}_{\mathcal{Q}}[R]\,\mathcal{I}_{\mathcal{Q}}[v]\big) = \mathcal{F}_{\mathcal{S}}[R]\,\mathcal{I}_{\mathcal{S}}[v], \\ \mathcal{F}_{\mathcal{Q}}[R](0)\,\mathcal{I}_{\mathcal{Q}}[v](0) = C_0, \end{cases} \quad \text{where} \quad \begin{aligned} \mathcal{I}_{\mathcal{Q}}[v](t) &= \int_\Omega \mathcal{K}^v_{\mathcal{Q}}[v](\mathbf{x},t)\,\mathrm{d}\mathbf{x}, \\ \mathcal{I}_{\mathcal{S}}[v](t) &= \int_\Omega \mathcal{K}^v_{\mathcal{S}}[v](\mathbf{x},t)\,\mathrm{d}\mathbf{x}. \end{aligned} \tag{2.19}$$

Here, $\mathcal{I}_{\mathcal{Q}}, \mathcal{I}_{\mathcal{S}} : (\Omega \times [0,T] \to \mathbb{R}) \to ([0,T] \to \mathbb{R})$ are the integration operators of the renormalized integration kernels $\mathcal{K}^v_{\mathcal{Q}}$, $\mathcal{K}^v_{\mathcal{S}}$. Thus, by alternately solving Eq. Eq. (2.18), the TDSR method guarantees that the solutions adhere to intrinsic physical properties. This framework holds for both conservative and dissipative systems, and allows a flexible integration of various PDE systems. In the context of traditional numerical methods, the structure ODE Eq. (2.19) is solved either by deriving the analytical solution [17] or by fixed-point iteration [40]. TDSR ensures theoretical guarantees for structure-preserving properties and has been effectively applied to a variety of PDE systems.

Example 2.4 (Example 2.1 revisited)**.** For the Burgers' equation (2.9), we introduce a time-dependent factor $R(t)$ to satisfy the structure equation (2.10). By setting $u(x,t) = R(t)\,v(x,t)$, the whole system is rewritten as

$$\begin{cases} R\,v_t + R_t\,v + R^2 v v_x - \nu R\,v_{xx} = 0, \\ \dfrac{\mathrm{d}}{\mathrm{d}t}\mathcal{E}_B[R\,v] = \mathcal{S}_B[R\,v]. \end{cases} \tag{2.20}$$

where $R(0) = 1$, $v(x,0) = u_0(x)$, and $\mathcal{E}_B[R\,v](0) = \int_{-1}^{1} u_0^2(x)\,\mathrm{d}x$. Following the previous discussion to substrate $R(t)$ from the integration kernel, it can be further simplified as

$$\mathcal{E}_B[R\,v](t) = R^2(t)\int_{-1}^{1} v^2(x,t)\,\mathrm{d}x, \quad \mathcal{S}_B[R\,v](t) = -2\nu R^2(t)\int_{-1}^{1} v_x^2(x,t)\,\mathrm{d}x.$$

Then the structure equation (2.10) can be rewritten into an ODE for $R(t)$ as

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\big(R^2\,\mathcal{I}_{\mathcal{Q}}[v]\big) = -2\nu R^2\,\mathcal{I}_{\mathcal{S}}[v], \\ R^2(0)\,\mathcal{I}_{\mathcal{Q}}[v](0) = \displaystyle\int_{-1}^{1} u_0^2(x)\ \mathrm{d}x. \end{cases} \quad \text{where} \quad \begin{aligned} \mathcal{I}_{\mathcal{Q}}[v](t) &= \int_{-1}^{1} v^2(x,t)\ \mathrm{d}x; \\ \mathcal{I}_{\mathcal{S}}[v](t) &= \int_{-1}^{1} v_x^2(x,t)\ \mathrm{d}x. \end{aligned} \tag{2.21}$$

Therefore, for a given solution $\bar{v}(x,t)$ from the primary solver, the TDSR factor $\bar{R}(t)$ can be obtained by solving the ODE Eq. (2.21), leading to the approximate solution $\bar{u}(x,t) = \bar{R}(t)\,\bar{v}(x,t)$.

Chapter 3

NNs Trained by Weight Permutation are Universal Approximators

In this chapter, we provide a theoretical foundation for a novel training method known as *permutation training*. Specifically, we prove that a permutation-trained ReLU MLP with random initializations can achieve UAP for any one-dimensional continuous function. Our numerical experiments confirm these theoretical findings by demonstrating the UAP's prevalence across various initialization strategies. Additionally, the experiments highlight the significant impact of initialization on permutation training performance and reveal intriguing patterns that suggest its potential for describing learning behaviors [6]. Our main findings are summarized below:

1. We prove the UAP of permutation-trained ReLU networks with pairwise random initialization to one-dimensional continuous functions.

2. The numerical experiments of regression problems emphasize the crucial role played by the initializations in the permutation training scenario.

3. By observing the permutation patterns, we find that permutation training as a new approach holds promise in describing intricate learning behaviors.

3.1 Introduction

In this section, we briefly discuss the related works of achieving UAP via permutation training, which serves as a complement to the introduction in Section 1.1.

3.1.1 Related works

Permutation is a typical group structure with a systematic description [8]. In deep learning, it closely relates to permutation equivariant or invariant networks [16] designed to learn from symmetrical data [55, 91]. It is also evident in graph-structured data, which inherently exhibits permutation invariance [65, 75]. However, permutation training is not limited to the issues with intrinsic symmetry.

As for the weight permutation attempts, [71] empirically proposed the first (to our knowledge) weight-permuted training method. This method preserves the initialized weight value, allowing more efficient and reconfigurable implementation of the physical neural networks [48, 49]. Our work provides theoretical guarantees of this method and considers some regression tasks numerically. Additionally, initialization can be improved by rewiring neurons from the perspective of computer networks [76], but the training methods are unchanged.

Permutation training is also closely related to the permutation symmetry and Linear Mode Connectivity (LMC) [22, 27]. The LMC suggests that after a proper permutation, most stochastic gradient descent solutions under different initialization will fall in the same basin in the loss landscape. Similarly, permutation training also seeks a permutation to improve performance. Therefore, the search algorithms utilized in LMC indicate the possibility of more efficient permutation training algorithms, as the fastest algorithm can search a proper permutation of large ResNet models in seconds to minutes [1, 44].

3.1.2 Outline

We state the main result and proof ideas in Section 3.2. In Section 3.3, we provide a detailed construction of the proof. The numerical results of permutation training are presented in Section 3.4, along with the observation of permutation behavior during the training process.

3.2 Main Results

This section introduces the UAP theorems, accompanied by a brief discussion of the proof idea.

3.2.1 Architecture of Shallow NNs

We start with an MLP in Eq. (2.1) equipped with ReLU activation functions and a single hidden layer with N hidden neurons. It has the form of a linear combination of ReLU basis functions like

$$f(x) = \sum_{i=1}^{N} a_i \mathrm{ReLU}(w_i x + b_i) + c, \quad \mathrm{ReLU}(z) = \max\{z, 0\},$$

where all parameters are scalars when approximating one-dimensional functions, *i.e.*, $w_i, b_i, a_i, c \in \mathbb{R}$. Since ReLU activation is positively homogeneous *i.e.*, $\mathrm{ReLU}(\lambda x) = \lambda \mathrm{ReLU}(x)$ for all $\lambda > 0$, we consider a homogeneous case with $w_i = \pm 1$. To facilitate our construction below, we assume N is an even number (*i.e.*, $N = 2n$) and the basis functions located pairwisely as

$$\phi_k^{\pm}(x) = \mathrm{ReLU}\big(\pm(x - b_k)\big), \quad k = 1, 2, ..., n, \tag{3.1}$$

where the biases $\{b_k\}_{k=1}^n$ determine the basis locations. Next, we introduce two factors α, γ to adjust the network's output, leading to an additional one-dimensional linear layer. While this layer is not essential for achieving UAP, it does simplify the

proof and offer practical value. The network's output function f^{NN} gives

$$f^{\mathrm{NN}}(x) = \alpha + \gamma \sum_{k=1}^{n} \left[p_k \phi_k^+(x) + q_k \phi_k^-(x) \right], \tag{3.2}$$

where $\boldsymbol{\theta}^{(2n)} = (p_1, q_1, ..., p_n, q_n) \in [-1, 1]^{2n}$ are the coefficient vector of basis functions, which also correspond to the parameters in the second hidden layer of the network.

Remark 3.1. The requirement of even N can be removed by adding a *ghost basis function* $\phi_k^{\pm}(x)$ with $a_i = 0$ if it can be paired with an existing $\phi_k^{\mp}(x)$ also with $a_i = 0$. Nevertheless, we retain this condition for simplicity.

3.2.2 Weight Configuration and Main Theorems

Without loss of generality, we consider the target continuous function $f^* \in C([0, 1])$. The permutation training is applied on the weights $\boldsymbol{\theta}^{(2n)}$ within second hidden layer's, leading to the following configuration: the coefficient vector $\boldsymbol{\theta}^{(2n)}$ is permuted from a predetermined vector $\mathbf{w}^{(2n)} \in \mathbb{R}^{2n}$, *i.e.*, $\boldsymbol{\theta}^{(2n)} = \tau(\mathbf{w}^{(2n)})$. To aid in readers' comprehension, we begin with a simple scenario with equidistantly distributed location vector $\mathbf{b}_{\mathrm{equi}}^{(n)}$ and pairwise coefficient vector $\mathbf{w}_{\mathrm{equi}}^{(2n)}$ as

$$\begin{aligned} \mathbf{b}_{\mathrm{equi}}^{(n)} &= (b_i)_{i=1}^n := \left(0, \tfrac{1}{n-1}, \cdots, 1\right), \\ \mathbf{w}_{\mathrm{equi}}^{(2n)} &= (\pm b_i)_{i=1}^n := (+b_1, -b_1, ..., +b_n, -b_n). \end{aligned} \tag{3.3}$$

The UAP of a permutation-trained network f^{NN} in Eq. (3.2) to f^* can be stated as follows:

Theorem 3.2 (UAP with a linear layer)**.** *For any function $f^* \in C([0, 1])$ and any small number $\varepsilon > 0$, there exists a large integer $n \in \mathbb{Z}^+$, and $\alpha, \gamma \in \mathbb{R}$ for f^{NN} in Eq. (3.2) with equidistantly distributed $\mathbf{b}_{equi}^{(n)}$ and $\mathbf{w}_{equi}^{(2n)}$ in Eq. (3.3), along with a permuted coefficients $\boldsymbol{\theta}^{(2n)} = \tau(\mathbf{w}_{equi}^{(2n)})$, such that $|f^{NN}(x) - f^*(x)| \leq \varepsilon$ for all $x \in [0, 1]$.*

The intuition of this result comes from the rich expressive possibilities of permutation training. Next, we enhance the result in Theorem 3.2 to a purely permuted situation, suggesting the UAP can be achieved without changing α, γ in Eq. (3.2).

Theorem 3.3 (UAP without the linear layer)**.** *Let $\alpha = 0, \gamma = 1$. For any function $f^* \in C([0,1])$ and any small number $\varepsilon > 0$, there exists a large integer $n \in \mathbb{Z}^+$, for f^{NN} in Eq. (3.2) with equidistantly distributed $\mathbf{b}_{equi}^{(n)}$ and $\mathbf{w}_{equi}^{(2n)}$ in Eq. (3.3), along with a permuted coefficients $\boldsymbol{\theta}^{(2n)} = \tau(\mathbf{w}_{equi}^{(2n)})$ such that $|f^{NN}(x) - f^*(x)| \leq \varepsilon$ for all $x \in [0,1]$.*

Although Theorem 3.3 considers a theoretically stronger setting, the additional requirement of n reveals the practical meanings of learnable α, γ in reducing the necessary network width to achieve UAP. Moreover, the result can be generalized to the scenario with pairwise random initialization:

$$\mathbf{b}_{\text{rand}}^{(n)} \sim \mathcal{U}[0,1]^n, \quad \mathbf{w}_{\text{rand}}^{(2n)} = (\pm p_i)_{i=1}^n, \ (p_i)_{i=1}^n \sim \mathcal{U}[0,1]^n. \tag{3.4}$$

The result is stated by the following theorem.

Theorem 3.4 (UAP for randomly initialized parameters)**.** *Given a probability threshold $\delta \in (0,1)$, for any function $f^* \in C([0,1])$ and any small number $\varepsilon > 0$, there exists a large integer $n \in \mathbb{Z}^+$, and $\alpha, \gamma \in \mathbb{R}$ for f^{NN} in Eq. (3.2) with randomly initialized $\mathbf{b}_{rand}^{(n)}$ and $\mathbf{w}_{rand}^{(2n)}$ in Eq. (3.4), along with a permuted coefficients $\boldsymbol{\theta}^{(2n)} = \tau(\mathbf{w}_{rand}^{(2n)})$, such that with probability $1 - \delta$, $|f^{NN}(x) - f^*(x)| \leq \varepsilon$ for all $x \in [0,1]$.*

3.2.3 Proof Outline

To establish the UAP of f^{NN} in Eq. (3.2) for $f^* \in C([0,1])$, we first approximate the target function f^* by a piecewise constant function g, leveraging a common approach in continuous function approximation [82]. Since g can be expressed as a sum of step functions, we can construct a subnetwork within f^{NN} to approximate

each individual step function. Finally, we need to eliminate the remaining part of f^{NN} to ensure that the approximation error is controlled. In this spirit, our constructive proof includes the following three steps (illustrated in Figure 3.1):

1. Approach the target function f^* by a piecewise constant function g;
2. Approximate each step functions of g by a subnetwork within f^{NN};
3. Annihilate the impact of the remaining parts of f^{NN}.

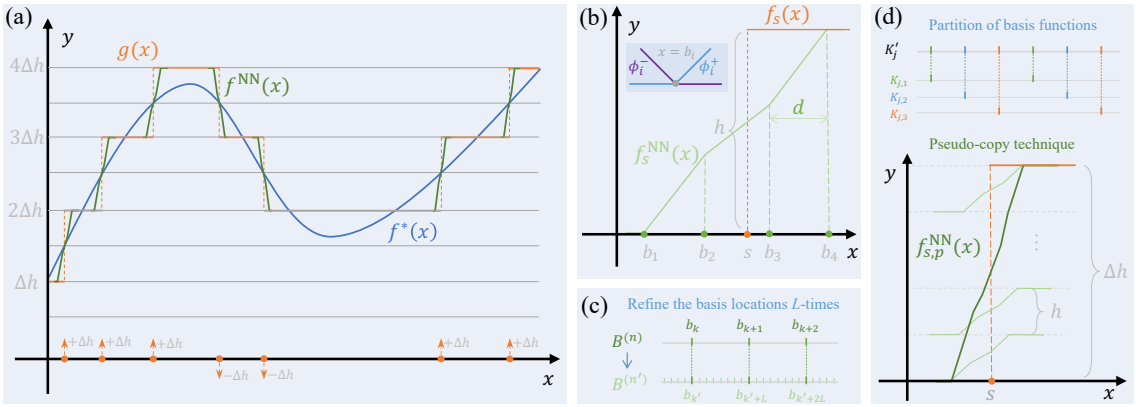

Figure 3.1: Main idea of the construction. (a) Approximate the continuous function f^* by a piecewise constant function g which is further approximated by permuted networks f^{NN}. (b) The step function approximator f_s^{NN} is constructed by step-matching. (c) Refine the basis functions L-times. (d) Stacking pseudo-copies to achieve the desired height.

Step 1 can be achieved by cutting the range of f^* into subregions with an equal width Δh, and then constructing a step function at each point where f^* crosses a boundary of these subregions, leading to an approximator with an error Δh (illustrated in Figure 3.1(a)). The detailed construction is given in Lemma 3.5.

The execution of step 2 is inspired by the divide-and-conquer algorithm in computer science [38]. For each step function f_{s_j} in g, we select basis functions to construct a step function approximator $f_{s_j}^{\mathrm{NN}}$, then sum them up to approach g. This *step-matching* construction utilizes four pairs of basis functions $\{\pm b_i\}_{i=1}^4$ (shown in Figure 3.1(b)), and establishing a one-to-one mapping between coefficients and

biases, *i.e.*, $\{p_i, q_i\}_{i=1}^4 = \{\pm b_i\}_{i=1}^4$. It ensures that each coefficient and location is uniquely assigned and prevents conflict between different approximators. The detailed construction is discussed in Section 3.3.2.

Step 3 plays a vital role in the proof construction, serving as an essential distinguishing factor that sets permutation training apart from conventional scenarios. Note that the specific setting of permutation training poses a crucial challenge that the proof must utilize every parameter, rather than just pick up the desired parameters and discard the rest. Therefore, after the construction in step 2, it is essential to eliminate the remaining network parameters to prevent the potential accumulation of errors. We solve this problem by proposing a *linear reorganization* to write the remaining part as a linear function with a controllable slope. The detailed reorganization is given in Section 3.3.3. Combining the above three steps, we can achieve the UAP of f^{NN} in Eq. (3.2) for any $f^* \in C([0,1])$. The whole proof of Theorem 3.2 is presented in Section 3.3.4.

To further enhance the conclusion of Theorem 3.2 to Theorem 3.3, we first introduce a technique called *pseudo-copy*, which can achieve UAP without altering γ in Eq. (3.2). By partitioning the refined basis functions, several pseudo-copies $f_{s,p}^{\mathrm{NN}}$ of the original approximator f_s^{NN} can be constructed with a controllable error. The final height can then be achieved by stacking these copies together rather than changing γ (see Figure 3.1(d)). Additionally, to make the shift factor α removable, our *constant-matching* construction can provide the necessary shifting. It also enables another way to eliminate the remaining part of the network. The detailed proof of Theorem 3.3 is given in Section 3.3.5.

Extending the UAP to the random initializations in Theorem 3.4 is justified by the fact that the parameters randomly sampled from uniform distributions become denser, thus approaching the equidistant case. Therefore, a sufficiently wide network has a high probability of finding a subnetwork that is close enough to the network with UAP in the equidistant case. Then this subnetwork can also achieve UAP due to its continuity. The remaining part of the network can be eliminated by step 3.

The detailed proof of Theorem 3.4 is given in Section 3.3.7.

3.3 UAP of Permutation-Trained NNs

This section first provides a detailed construction of the approximator with a weight-permuted NN in the equidistant case, along with an estimation of the convergent rate of approximation error. The extension to the scenario with random initialization is also thoroughly discussed.

3.3.1 Approximate the Target Function with a Piecewise Constant Function

As we discussed previously, the first step of our proof is to approximate the target function f^* by a piecewise constant function g. It can be summarized as the following lemma, which is based on the Stone-Weierstrass theorem [83].

Lemma 3.5. *For any function $f^* \in C([0,1])$ and any small number $\varepsilon' > 0$, there is a piecewise constant function g with a common jump $\Delta h \leq \varepsilon'$, such that $|g(x) - f^*(x)| \leq \varepsilon'$ for all $x \in [0,1]$. Moreover, the function g can be written as a summation of J step functions $\{f_{s_j}\}_{j=1}^J$ as the following form,*

$$g(x) = \sum_{j=1}^{J} a_j f_{s_j}(x) = \sum_{j=1}^{J} a_j \Delta h\, \chi(x - s_j), \quad a_j = \pm 1,\ s_j \in [0,1],\ J \in \mathbb{Z}^+. \tag{3.5}$$

Here J is the step number, s_j is the step location, a_j is the step sign controlling the direction, and χ is the standard step function

$$\chi(x) = \begin{cases} 0, & x < 0, \\ 1, & x \geq 0. \end{cases}$$

Proof. For a given target function f^*, the function g can be constructed explicitly.

Thanks to the Stone-Weierstrass theorem [83], we assume f^* to be a polynomial function for simplicity. Let the range of f^* be covered by an interval $[k_{\min}\Delta h, k_{\max}\Delta h]$ with two integers $k_{\min}, k_{\max} \in \mathbb{Z}$. Then denote J as the number of intersections between f^* and parallel lines $y = (k + 0.5)\Delta h$, $k = k_{\min}, k_{\min} + 1, \cdots, k_{\max} - 1$. Notice that J must be finite since f^* is a polynomial function (Otherwise f^* will be a constant function due to the fundamental theorem of algebra [43], which can be directly approximated by g). The scenario with $J = 0$ is also trivial since it indicates that f^* lies in a single interval $[(k_0 - 0.5)\Delta h, (k_0 + 0.5)\Delta h]$ for some integer $k_0 \in [k_{\min}, k_{\max}]$, such that f^* can be approached by the constant function $y = k_0\Delta h$ with an error Δh.

Hence, for any $j = 1, \cdots, J$, we choose $s_j \in [0, 1]$ such that $f^*(s_j) = (k_j + 0.5)\Delta h$ for some $k_j \in \mathbb{Z}$. The step locations $s_1 < \cdots < s_J$ are distinct since any repetition would contradict the continuity of f^*. The step sign a_j is determined according to the values of f^* on $[s_{j-1}, s_{j+1}]$. It is easy to verify that such a construction satisfies our requirements. □

3.3.2 Construct Step and Constant Function Approximators

Lemma 3.5 enables us to approximate the target function f^* by a piecewise constant function g in Eq. (3.5). Next, we aim to approximate each step function within g by a subnetwork of f^{NN}, respectively, and then eliminate the remaining part of f^{NN}. This section introduces several key constructions to achieve this goal in the equidistant case.

Step-Matching Construction of Step Function Approximators f_s^{NN}

Here we construct the step function approximator f_s^{NN} for a given step function $f_s(x) = \Delta h\, \chi(x - s)$ with height Δh and location s. The construction considers four pairs of basis functions $\{\phi_i^{\pm}\}_{i=1}^4$ with locations $\{b_i\}_{i=1}^4$ and coefficients $\{p_i, q_i\}_{i=1}^4 =$

$\{\pm b_i\}_{i=1}^4$, which has the following form,

$$f_s^{\mathrm{NN}}(x) = \sum_{i=1}^4 p_i \phi_i^+(x) + \sum_{i=1}^4 q_i \phi_i^-(x), \quad x \in [0,1]. \tag{3.6}$$

Here we require the locations $b_1 < b_2 < b_3 < b_4$ satisfy the following symmetric condition,

$$d := b_2 - b_1 = b_4 - b_3 \quad \Longrightarrow \quad b_1 + b_4 = b_2 + b_3, \tag{3.7}$$

where d is the basis distance. To ensure a local error of the approximator, we appeal f_s^{NN} to be x-independent outside the interval $[b_1, b_4]$. As a result, the coefficients p_i, q_i must satisfy $\sum_{i=1}^4 p_i = \sum_{i=1}^4 q_i = 0$, which implies the correspondence between $\{p_i, q_i\}_{i=1}^4$ and $\{\pm b_i\}_{i=1}^4$ as

$$\begin{aligned} p_1 = -b_1, \quad p_2 = +b_2, \quad p_3 = +b_3, \quad p_4 = -b_4, \\ q_1 = +b_4, \quad q_2 = -b_3, \quad q_3 = -b_2, \quad q_4 = +b_1. \end{aligned} \tag{3.8}$$

We call the $\{p_i, q_i\}_{i=1}^4$ and $\{\pm b_i\}_{i=1}^4$ is *step-matching* if they satisfy Eq. (3.8), which gives the piecewise form of f_s^{NN} in Eq. (3.6) as

$$f_s^{\mathrm{NN}}(x) = \begin{cases} 2b_1 b_4 - 2b_2 b_3 & 0 \le x < b_1, \\ (-b_1 + b_4)x + b_1^2 + b_1 b_4 - 2b_2 b_3 & b_1 \le x < b_2, \\ (-2b_1 + 2b_2)x + b_1^2 - b_2^2 + b_1 b_4 - b_2 b_3 & b_2 \le x < b_3, \\ (-b_1 + b_4)x + b_1^2 - b_2^2 - b_3^2 + b_1 b_4 & b_3 \le x < b_4, \\ b_1^2 - b_2^2 - b_3^2 + b_4^2 & b_4 \le x \le 1. \end{cases} \tag{3.9}$$

The profile of this f_s^{NN} can be found in Figure 3.1(b), which shows that f_s^{NN} is monotone and can approach a step function with the height h satisfying the following relation,

$$h = 2(b_1^2 - b_2^2 - b_3^2 + b_4^2) = 4d(b_4 - b_2). \tag{3.10}$$

Notice that choosing the basis functions adjacent will lead to $d = \frac{1}{n-1}$ and $h = 8d^2$. By shifting $h/2$ and scaling $\Delta h/h$, we use f_s^{NN} to approach step function f_s with $s \in [b_1, b_4]$. It is obvious that the L^∞ error has the following trivial bound,

$$\left| \frac{\Delta h}{h} \left[f_s^{\mathrm{NN}}(x) + \frac{h}{2} \right] - f_s(x) \right| \leq \Delta h, \quad \forall\, x \in [0,1]. \tag{3.11}$$

Notice that Eq. (3.9) implies that the approximation is exactly accurate when $x \notin [b_1, b_4]$. A toy example of this step-matching construction is shown in Figure 3.2.

Remark 3.6. Although [7] suggested that a step function could be approached by two ReLU basis functions, this approach is unsuitable for the permutation training scenario. For an error tolerance ε, a step function $f_s(x) = h_s \chi(x - s)$ can be well approximated by a linear combination of two ReLU basis functions as

$$\widetilde{f}_s^{\mathrm{NN}}(x) = \frac{h_s}{2\varepsilon}[\mathrm{ReLU}(x - s + \varepsilon) - \mathrm{ReLU}(x - s - \varepsilon)].$$

However, the dependence of the coefficients on the step height h_s hinders further construction, as the permutation training scenario can use each coefficient only once.

Example 3.1. To illustrate the key idea of our construction, we consider a toy example of approaching a step function $f_s(x) = 0.8\,\chi(x - 0.4)$ for $x \in [0,1]$ by a network in Eq. (3.2) with $n = 11$ and equidistantly initialized $\{b_k\}_{k=1}^{11} = \{0, 0.1, \cdots, 1\}$. This setting is considered after applying Lemma 2.1 with $\Delta h = 0.8$.

Following our step-matching construction, we can choose the basis functions with $\{b_k\}_{k=1}^{4} = \{0.1, 0.3, 0.6, 0.8\}$ to be step-matching in Eq. (3.8), leading to a step function approximator f_s^{NN}. The height of f_s^{NN} is given by $h = 0.4$. Therefore, the target step function f_s can be approximated by f_s^{NN} along with shifting $\Delta h/2 = 0.4$ and scaling $\Delta h/h = 2$ with the following error estimation:

$$\left| \left[\frac{\Delta h}{h}(x)\, f_s^{\mathrm{NN}} + \frac{h}{2} \right] - f_s(x) \right| \leq 0.8 = \Delta h, \quad x \in [0,1].$$

The target step function f_s and its approximator f_s^{NN} are plotted in Figure 3.2.

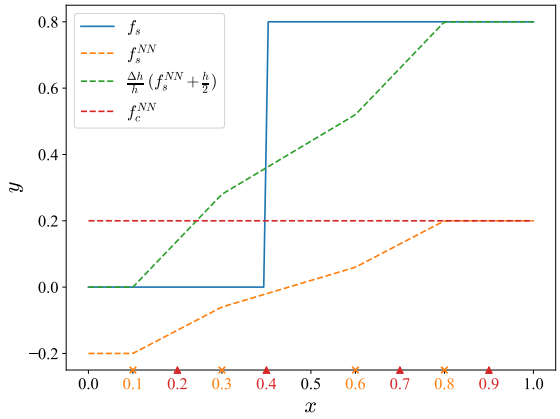

Figure 3.2: Approximating a step function $f_s(x) = 0.8\,\chi(x - 0.4)$ and a constant function $f_c(x) = 0.2$ in $x \in [0, 1]$ with the step-function approximator f_s^{NN} and the constant function approximator f_c^{NN}, respectively. The target functions are plotted as lines, and the approximation results are shown as dashed lines. The locations of the basis functions used for f_s^{NN} and f_c^{NN} are marked with "$\times$" and "▲" on the x-axis, respectively.

Constant-Matching Construction of Constant Function Approximators $f_c^{\mathbf{NN}}$

The four-pair form in Eq. (3.6) can also be utilized to approximate a constant function, which plays a crucial role in the proof of Theorem 3.3. The constant function approximator (denoted as f_c^{NN}) shares the same form as f_s^{NN} in Eq. (3.6) but differs slightly in the parameter assignment in Eq. (3.8). The coefficients $\{p_i, q_i\}_{i=1}^4$ are set to equalize the height of the two constant pieces $x < b_1$ and $b_4 \le x$, leading to $-\sum_{i=1}^4 p_i b_i = \sum_{i=1}^4 q_i b_i$. A possible choice is

$$\begin{aligned} p_1 &= -b_1, \quad p_2 = +b_2, \quad p_3 = +b_3, \quad p_4 = -b_4, \\ q_1 &= +b_1, \quad q_2 = -b_2, \quad q_3 = -b_3, \quad q_4 = +b_4. \end{aligned} \tag{3.12}$$

We call the $\{p_i, q_i\}_{i=1}^4$ and $\{\pm b_i\}_{i=1}^4$ is *constant-matching* if they satisfy Eq. (3.12). With these chosen coefficients $\{p_i, q_i\}_{i=1}^4$ and the the symmetry of coefficients in

Eq. (3.7), the constant function approximator f_c^{NN} can be written as:

$$\begin{aligned} f_c^{\mathrm{NN}}(x) &= (b_2 + b_3 - b_1 - b_4)\, x + b_1^2 - b_2^2 - b_3^2 + b_4^2 \\ &= 2d(b_4 - b_2) = \frac{h}{2}, \quad \forall\, x \in [0,1]. \end{aligned} \tag{3.13}$$

By the relations of h in Eq. (3.10), it gives a representation of constant $C = h/2$ without the approximation error, i.e., $|f_c^{\mathrm{NN}}(x) - h/2| = 0$ for all $x \in [0,1]$.

Furthermore, by changing the sign of $\{p_i, q_i\}_{i=1}^4$ in Eq. (3.12) simultaneously, an approximator f_{-c}^{NN} with a negative constant $-C = -h/2$ can also be constructed:

$$f_{-c}^{\mathrm{NN}}(x) = -2d(b_4 - b_2) = -\frac{h}{2}, \quad \forall\, x \in [0,1].$$

Therefore, we can freely adjust the sign $a_j = \pm 1$ of the approached constant. It also enables us to construct f_c^{NN} and f_{-c}^{NN} with the same constant separately, then pair them up to offset with each other, *i.e.*, $f_c^{\mathrm{NN}}(x) + f_{-c}^{\mathrm{NN}}(x) = 0$ for all $x \in [0,1]$. A toy example of this constant-matching construction is shown in Figure 3.2.

Example 3.2 (Example 3.1 continued)**.** In the previous example, we can also consider the constant function approximator f_c^{NN} following our constant-matching construction in Eq. (3.12). We choose the basis functions located at $\{0.2, 0.4, 0.7, 0.9\}$ to be constant-matching, leading to a constant function approximator $f_c^{\mathrm{NN}} = h/2$. The height of f_c^{NN} is given by $h/2 = 0.4$. From the shape of f_c^{NN} plotted in Figure 3.2, we can see that it can approximate a constant function without any error.

3.3.3 Annihilate the Unused Part of NNs

After constructing approximators to approach the target function, the permutation training setting requires that the remaining parameters be suitably arranged to eliminate their impact. Notice that a pair of basis functions $\phi_i^{\pm}$ are either used together or not at all. Concretely, we choose a pair of basis functions located at b_i and set the coefficients as $\{p_i, q_i\} = \{\pm b_i\}$. The linear function approximator f_ℓ^{NN}

can be written as the following form,

$$f_\ell^{\mathrm{NN}}(x) = m_i \ell_i, \quad \ell_i(x) := b_i \phi_i^+(x) - b_i \phi_i^-(x) = b_i x - b_i^2, \quad x \in [0,1], \tag{3.14}$$

where $m_i = \pm 1$ is a freely adjusted sign. We call this method *linear reorganization*. It allows us to adjust the number of the total basis functions, as the L^∞-norm of f_ℓ^{NN} located at b_i has an upper bound of b_i^2. Hence, we can choose the unwanted b_i that is small enough, then omit it along with the related coefficients $\{p_i, q_i\} = \{\pm b_i\}$ from the network without affecting the approximation error.

Concretely, denote $I_{\mathrm{un}} \subset \{1, 2, ..., n\}$ as the index of unused basis functions and $\bar{n}$ as the number of elements in I_{un}, which can be processed to be an even number. After taking a sum of all $i \in I_{\mathrm{un}}$, the resulting $\mathcal{S}_\ell$ has the following linear form,

$$\mathcal{S}_\ell(x) = \sum_{i \in I_{\mathrm{un}}} f_{\ell_i}^{\mathrm{NN}}(x) = \sum_{i \in I_{\mathrm{un}}} m_i b_i x - \sum_{i \in I_{\mathrm{un}}} m_i b_i^2 =: \beta x + \eta, \quad x \in [0,1], \tag{3.15}$$

where β is the slope and η is the intercept. The goal then is to control the L^∞-norm of $\mathcal{S}_\ell$. We first choose a proper $m_i \in \{-1, 1\}$ for each $i \in I_{\mathrm{un}}$ to reduce $|\beta|$, which is equivalent to assigning addition and subtraction operations within a given sequence to reduce the final result's absolute value. Motivated by the *Leibniz's test* (known as *alternating series test*) [73], the following lemma offers a solution with an upper bound of β.

Lemma 3.7. *For an even number $\bar{n}$ and a sequence of real number $c_i \in \mathbb{R}, i = 1, \cdots, \bar{n}$, there exists a choice of $m_i \in \{-1, 1\}$, $i = 1, \cdots, \bar{n}$, such that*

$$0 \le \sum_{i=1}^{\bar{n}} m_i c_i \le \Delta c, \quad \Delta c = \max_{\substack{1 \le j \le \bar{n} \\ j \notin \operatorname{argmax} c_j}} \left(\min_{\substack{i \ne j \\ c_i \ge c_j}} (c_i - c_j) \right), \tag{3.16}$$

where Δc can be regarded as the largest gap between the adjacent elements after sorting c_i in descending order.

Proof. Without loss of generality, we assume $c_1 \ge c_2 \ge \cdots \ge c_{\bar{n}} \ge 0$ and thus

$\Delta c = \max_i |c_i - c_{i-1}|$. Since $\bar{n}$ is an even number, we define a new series $\{r_j\}_{j=1}^{\bar{n}/2}$ as the difference between each pair of elements in $\{c_i\}_{i=1}^{\bar{n}}$,

$$r_j = c_{2j-1} - c_{2j} \geq 0, \quad j = 1, 2, \cdots, \frac{\bar{n}}{2}.$$

Then we permute $(r_j)_{j=1}^{\bar{n}/2}$ to be descending order. Concretely, we find a permutation τ and denote $r'_j = \tau(r_j) = [\tau(c_{2j-1}) - \tau(c_{2j})]$, such that $r'_1 \geq r'_2 \geq \cdots \geq r'_{\bar{n}/2} \geq 0$. Next, we alternatively use addition and subtraction operations on each r'_j by noting $\lambda_j = (-1)^{j-1}$, $j = 1, \cdots, \bar{n}/2$, and estimate the summation $\mathcal{S}_{\bar{n}/2}$ as the following forms,

$$\mathcal{S}_{\bar{n}/2} := \sum_{j=1}^{\bar{n}/2} \lambda_j r'_j = \begin{cases} (r'_1 - r'_2) + (r'_3 - r'_4) + \cdots \geq 0, \\ r'_1 - (r'_2 - r'_3) - (r'_4 - r'_5) - \cdots \leq r'_1 \leq \Delta c. \end{cases}$$

Note that $\mathcal{S}_{\bar{n}/2} = \sum_{j=1}^{\bar{n}/2} \lambda_j [\tau(c_{2j-1}) - \tau(c_{2j})]$ is of the form of $\sum_{i=1}^{\bar{n}} m_i c_i$, hence the choice of m_i implied by λ_j satisfies our requirement and the proof is finished □

Example 3.3. To illustrate the processing method of Lemma 3.7, we consider a given sequence

$$3,\ 1,\ 4,\ -1,\ 5,\ 9,$$

with the largest gap $\Delta c = 4$. The aim is to choose choose $m_i \in \{-1, 1\}$ to reduce the absolute value of the result

$$S = 3 \cdot m_1 + 1 \cdot m_2 + 4 \cdot m_3 - 1 \cdot m_4 + 5 \cdot m_5 + 9 \cdot m_6.$$

This can be achieved by the following steps:

1. Rearrange the sequence into 9, 5, 4, 3, 1, −1;
2. Compute the gap between every two adjacent elements as 4, 1, 2;
3. Reorganize the gap sequence into an alternating series and compute the result as $S = 4 - 2 + 1 = 3$;

4. Transfer the choice of $m_i \in \{-1, 1\}$ to the original sequence as

$$S = (9 - 5) - [1 - (-1)] + (4 - 3) = 9 - 5 - 1 - 1 + 4 - 3 = 3 < 4 = \Delta c.$$

Therefore, the result $S = 3$ can be achieved by choosing $m_1 = -1, m_2 = -1, m_3 = 1, m_4 = 1, m_5 = -1, m_6 = 1$.

Remark 3.8. Leibniz's test guarantees that an alternating series that decreases in absolute value is bounded by the leading term. However, we emphasize that utilizing Leibniz's test directly to $\{c_i\}_{i=1}^{\bar{n}}$ can only get a trivial bound. Therefore, the introduction of r_j is important to get the desired result.

After applying Lemma 3.7 to control the slope β, the intercept η is consequently determined by the chosen $\{m_i\}_{i \in I_{\text{un}}}$. Thus, we can choose a constant to adjust the intercept, which establishes an upper bound for the error S_ℓ of the remaining part.

Remark 3.9. We follow the conventional setting of the UAP study and primarily focus on shallow MLPs, as the generalization of the results to deep networks is expected. The linear reorganization in Eq. (3.14) enables construct an identity function $y = x$ using a pair of basis functions $y = p_n \phi_1^+(x) + q_n \phi_1^-(x)$, where $b_1 = 0$, $p_n = 1$, $q_n = -1$. This process enables us to utilize identity functions within subsequent layers. Consequently, the deep networks scenario parallels the shallow cases.

3.3.4 Proof of Theorem 3.2

Lemma 3.5 offers a piecewise constant function g to approach the target function f^*. Now we prove that by permuting the selected coefficients and eliminating the remaining part, f^{NN} can approximate the piecewise constant function g with the same accuracy, enabling us to prove Theorem 3.2.

Proof of Theorem 3.2. For any target function $f^* \in C([0, 1])$ and small number ε,

we aim to construct a network f^{NN} in Eq. (3.2) with n pairs of basis functions to approximate f^* with error ε. The goal is achieved with the following points:

a) Approach f^* by a piecewise constant function g. Employing Lemma 3.5 by letting $\varepsilon' = \varepsilon/4$, we can construct a piecewise constant function g in Eq. (3.5) with a constant height $\Delta h \leq \varepsilon' = \varepsilon/4$, distinct step locations $s_1 < \cdots < s_J$. It gives $|g(x) - f^*(x)| \leq \Delta h < \varepsilon/2$ for all $x \in [0,1]$. Here we denote $\delta_s := \max_j |s_j - s_{j-1}|$ as the maximal gap of step locations, where $j = 0, 1, \cdots, J+1$ and $s_0 = 0$, $s_{J+1} = 1$.

b) Approximate each step function in g by the step-matching approximator $f_{s_j}^{\mathrm{NN}}$ in Eq. (3.6). Since the step locations $\{s_j\}_{j=1}^J$ are distinct, we can choose a network f^{NN} with large enough $\hat{n}$, *i.e.*, the locations in $\mathbf{b}_{\mathrm{equi}}^{(\hat{n})}$ are dense enough, such that there are enough basis functions to construct $f_{s_j}^{\mathrm{NN}}$ for each $a_j f_{s_j}$, respectively. In fact, for any $\hat{n} > 8/\delta_s + 1$, the distance between basis functions $\hat{d} = 1/(\hat{n}-1)$ satisfies $\hat{d} < \delta_s/8$.

Additionally, to maintain the consistency of the following estimation, we refine the basis locations $\mathbf{b}_{\mathrm{equi}}^{(\hat{n})}$ to $\mathbf{b}_{\mathrm{equi}}^{(n)}$ with $n = L(\hat{n}-1)+1$ for some integer $L \geq \Delta h(\hat{n}-1)^2/8$ (see Figure 3.1(c)). Denote $K = \{1, \cdots, n\}$ as the index set of the locations $\mathbf{b}_{\mathrm{equi}}^{(n)}$, and for each $j = 1, \cdots, J$, we choose the basis functions with the index $K_j := \{k_j, k_j + L, k_j + 2L, k_j + 3L\} \subset K$, such that $s_j \in [b_{k_j+L}, b_{k_j+2L})$ and $\{K_j\}_{j=1}^J$ has empty intersection. Therefore, for each $a_j f_{s_j}$, an approximator $(f_{s_j}^{\mathrm{NN}} + a_j h/2)$ with a shifting $a_j h/2$ can be constructed by applying the step-matching construction on $\{p_k, q_k\}_{k \in K_j}$. This approximation has the following error estimation given by Eq. (3.11):

$$\left| \left[f_{s_j}^{\mathrm{NN}}(x) + a_j \frac{h}{2} \right] - a_j \frac{h}{\Delta h} f_{s_j}(x) \right| \leq h, \quad \forall\, x \in [0,1], \quad j = 1, \cdots, J, \tag{3.17}$$

where $h = 8\hat{d}^2 = 8/(\hat{n}-1)^2 = 8L^2/(n-1)^2$ is the height determined by $\hat{n}$, and the scaling $h/\Delta h$ serves to match the constant height Δh and its approximator height h. Our requirement of L gives the condition $L \geq \Delta h/h$. Furthermore, denote $K_{\mathrm{use}} = \cup_{j=1}^J K_j$ as the index set for all involved basis functions in Eq. (3.17), which

leads to the requirement of $\hat{n} > 4J$. We define $\mathcal{S}_{\text{use}}$ as the picked subnetwork of f^{NN},

$$\mathcal{S}_{\text{use}}(x) := \sum_{k\in K_{\text{use}}} \left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] = \sum_{j=1}^{J} f_{s_j}^{\text{NN}}(x), \quad x \in [0,1],$$

then the properties of our step-matching construction and Eq. (3.17) imply the error E_{use} of the summed approximators can be estimated in the following form,

$$\begin{aligned} E_{\text{use}} &:= \max_{x\in[0,1]} \left| \mathcal{S}_{\text{use}}(x) + \frac{h}{2}\sum_{j=1}^{J} a_j - \frac{h}{\Delta h} g(x) \right| \\ &= \max_{x\in[0,1]} \left| \sum_{j=1}^{J} \left[f_{s_j,p_l}^{\text{NN}}(x) + a_j\frac{h}{2} \right] - \sum_{j=1}^{J} \frac{h}{\Delta h} a_j f_{s_j}(x) \right| \le h. \end{aligned} \tag{3.18}$$

Here $\sum_{j=1}^{J} a_j =: J' \le J$ is a constant, and height h satisfies $h = 8L^2/(n-1)^2$.

c) Annihilate the impact of the unused part. The linear reorganization in Eq. (3.14) enables write the unused part in f^{NN} into a linear function $\mathcal{S}_{\text{un}}$, namely,

$$\mathcal{S}_{\text{un}}(x) = \sum_{k\in K\setminus K_{\text{use}}} \left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] = \sum_{k\in K\setminus K_{\text{use}}} (m_k b_k x - m_k b_k^2) =: \beta x + \eta.$$

Since the number of unused basis functions can be processed to be even, we apply Lemma 3.7 on the series $\{b_k\}_{k\in K\setminus K_{\text{use}}}$ to get a choice of $m_k \in \{-1, 1\}$ for each $k \in K \setminus K_{\text{use}}$, which provides an upper bound of the slope β as $0 \le \beta \le \Delta b$, where Δb is the largest gap between the adjacent locations in $\{b_k\}_{k\in K\setminus K_{\text{use}}}$. Notice that in f^{NN}, the refined basis distance $d < 1/L\hat{n}$ is small enough to ensure that there is at least one unused basis function between two adjacent used basis functions, *i.e.*, $\Delta b \le 2/L\hat{n}$. To control the error of the intercept η, we introduce a shifting $C_\eta = -\eta$, thus the error E_{un} introduced by the unused part gives the following estimation,

$$E_{\text{un}} := \max_{x\in[0,1]} \left|\mathcal{S}_{\text{un}}(x) + C_\eta\right| \le \Delta b < \frac{2}{L\hat{n}} \le \frac{2h}{\Delta h\hat{n}}. \tag{3.19}$$

Therefore, we choose $\hat{n} > 2/\Delta h$ such that the error E_{un} satisfies $E_{\text{un}} \le h$.

d) Complete the proof by choosing the suitable values of γ, α. We choose $\hat{n}$ and two factors γ, α, such that

$$\hat{n} \geq \max\left\{4J,\ \frac{8}{\delta_s}+1,\ \frac{2}{\Delta h}\right\}, \quad \gamma = \frac{\Delta h}{h}, \quad \alpha = \Delta h\left(\frac{J'}{2}+\frac{C_\eta}{h}\right).$$

Moreover, Let $n = L(\hat{n}-1)+1$ for some integer $L \geq \Delta h(\hat{n}-1)^2/8$, then Eq. (3.19)-(3.21) implies that the approximation error of the network f^{NN} with width n satisfies

$$\begin{aligned}
\left|f^{\mathrm{NN}}(x)-g(x)\right| &= \left|\gamma\sum_{k\in K}\left[p_k\phi_k^+(x)+q_k\phi_k^-(x)\right]+\alpha-g(x)\right| \\
&\leq \left|\frac{\Delta h}{h}\left[\mathcal{S}_{\mathrm{use}}(x)+\mathcal{S}_{\mathrm{un}}(x)\right]+\left(\frac{\Delta h}{2}J'+\frac{\Delta h}{h}C_\eta\right)-g(x)\right| \\
&\leq \frac{\Delta h}{h}\left|\mathcal{S}_{\mathrm{use}}(x)+\frac{h}{2}J'-\frac{h}{\Delta h}g(x)\right|+\frac{\Delta h}{h}\left|\mathcal{S}_{\mathrm{un}}(x)+C_\eta\right| \\
&\leq \frac{\Delta h}{h}(E_{\mathrm{use}}+E_{\mathrm{un}}) \leq 2\Delta h \leq \frac{\varepsilon}{2}, \quad \forall\, x\in[0,1].
\end{aligned}$$

This complete the proof since $|f^{\mathrm{NN}}(x)-f^*(x)| \leq \varepsilon$ for all $x \in [0,1]$. □

3.3.5 Proof of Theorem 3.3

Next, we remove scaling γ and shifting α during the proof, *i.e.,* achieve UAP with fixed factors $\gamma = 1, \alpha = 0$ as stated in Theorem 3.3. To eliminate the scaling γ, we introduce the *pseudo-copy* technique to let $\gamma = L$ for an integer $L \in \mathbb{Z}^+$, allowing copy the approximator L-times and stack them (instead of multiplying by γ) to match the desired height. The shifting α can be replaced by our constant-matching construction in Eq. (3.13), which is also applied to eliminate the remaining parameters, since our linear reorganization in Eq. (3.14) requires a shifting C_η to control the error.

Proof of Theorem 3.3. For a given target function $f^* \in C([0,1])$, we first apply Theorem 3.2 to obtain a network f^{NN} to approximate f^*, and then enhance the

construction by pseudo-copy technique and constant-matching construction to eliminate the scaling γ and shifting α, leading to a network f^{NN} in Eq. (3.2) with fixed factors $\gamma = 1, \alpha = 0$. To facilitate distinction, we will denote this network equipped with the pseudo-copy technique as $f_{\mathrm{pc}}^{\mathrm{NN}}$ in the following text.

a) Approximate the target function f^* by a network f^{NN} with learnable factors γ and α. By applying Theorem 3.2, we construct f^{NN} to approximate f^* through a piecewise constant function g with error ε. Following the previous discussion, we have $\Delta h \leq \varepsilon/8$, $\gamma = \Delta h/h$. During the construction, L is chosen to be $L = \gamma$.

b) Remove the scaling γ by the pseudo-copy technique. We first reassign the basis functions in f^{NN} to construct the pseudo-copies of each approximator $f_{s_j}^{\mathrm{NN}}$. For each $K_j = \{k_j, k_j + L, k_j + 2L, k_j + 3L\} \subset K$, we denote the corresponding adjacent index set for pseudo-copy as $K_j' := \{k_j, k_j + 1, \cdots, k_j + 4L - 1\}$ such that $K_j \subset K_j'$. To maintain the same height of each pseudo-copy, we partition K_j' into L subsets as $K_j' = \cup_{l=1}^{L} K_{j,l}$ by choosing after every L indexes (illustrated at the top of Figure 3.1(d)).

For each $l = 1, \cdots, L$, we apply the step-matching construction on $\{p_k, q_k\}_{k \in K_{j,l}}$ to construct the pseudo-copy $f_{s_j,p_l}^{\mathrm{NN}}$ for $f_{s_j}^{\mathrm{NN}}$ in Eq. (3.17), respectively. Since each $f_{s_j,p_l}^{\mathrm{NN}}$ has the same height h with $f_{s_j}^{\mathrm{NN}}$, we have the copy error as $|f_{s_j,p_l}^{\mathrm{NN}}(x) - f_{s_j}^{\mathrm{NN}}(x)| < h$ for all $x \in [0, 1]$. Therefore, Eq. (3.17) allows the summed $f_{s_j,p_l}^{\mathrm{NN}}$ to approximate $a_j f_{s_j}$ with the following error:

$$\begin{aligned} &\left| \sum_{l=1}^{L} \left[f_{s_j,p_l}^{\mathrm{NN}}(x) + a_j \frac{h}{2} \right] - a_j f_{s_j}(x) \right| \\ \leq &\sum_{l=1}^{L} \left| f_{s_j,p_l}^{\mathrm{NN}}(x) - f_{s_j}^{\mathrm{NN}}(x) \right| + \left| L \left[f_{s_j}^{\mathrm{NN}}(x) + a_j \frac{h}{2} \right] - a_j f_{s_j}(x) \right| \\ \leq &Lh + \Delta h = 2\Delta h, \quad \forall\, x \in [0, 1], \quad j = 1, \cdots, J. \end{aligned} \tag{3.20}$$

Here, $a_j h/2$ is the shifting required by the pseudo-copies.

c) Replace the shifting with a constant-matching construction. After constructing the pseudo-copies $f_{s_j,p_l}^{\mathrm{NN}}$, our constant-matching construction in Eq. (3.13) can

pair them up with the necessary shifting $f_{c_j}^{\mathrm{NN}} = a_j h/2$, enabling the combined $\sum_{l=1}^{L}(f_{s_j,p_l}^{\mathrm{NN}} + f_{c_j}^{\mathrm{NN}})$ to approach $a_j f_{s_j}$. Denote K'_{use} as the index set for all involved basis functions. We define $\mathcal{S}'_{\mathrm{use}}$ as the picked subnetwork of $f_{\mathrm{pc}}^{\mathrm{NN}}$,

$$\mathcal{S}'_{\mathrm{use}}(x) := \sum_{k\in K'_{\mathrm{use}}} \left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] = \sum_{l=1}^{L}\sum_{j=1}^{J}\left[f_{s_j,p_l}^{\mathrm{NN}}(x) + f_{c_j}^{\mathrm{NN}}(x)\right],$$

where $x \in [0,1]$. Eq. (3.20) implies the following error estimation,

$$\begin{aligned} E'_{\mathrm{use}} &:= \max_{x\in[0,1]} |\mathcal{S}'_{\mathrm{use}}(x) - g(x)| \\ &= \max_{x\in[0,1]} \left| \sum_{j=1}^{J}\sum_{l=1}^{L}\left[f_{s_j,p_l}^{\mathrm{NN}}(x) + f_{c_j}^{\mathrm{NN}}(x)\right] - \sum_{j=1}^{J} a_j f_{s_j}(x)\right| \le 2\Delta h. \end{aligned} \tag{3.21}$$

This construction uses $8L$ basis locations, leading to the requirement that $n > 8LJ$.

d) Eliminate the unused part of the network. Since the linear reorganization in Eq. (3.14) is invalid, we turn to apply the constant-matching construction in Eq. (3.13) to eliminate the unused part of the network. Concretely, for the remaining $n - 8JL$ pairs of basis functions in $f_{\mathrm{pc}}^{\mathrm{NN}}$, we construct $f_{\pm c_t}^{\mathrm{NN}}$, $t = 1, \cdots, \lfloor n/8\rfloor - JL$, then pairing them up to offset with each other, *i.e.*,

$$\mathcal{S}'_{\mathrm{un}}(x) = \sum_{k\in K\setminus K'_{\mathrm{use}}} \left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] = \sum_{t=1}^{\lfloor n/8\rfloor - JL} \left[f_{c_t}^{\mathrm{NN}}(x) + f_{-c_t}^{\mathrm{NN}}(x)\right] + R(x).$$

Here $R(x)$ is the residual part since n is not always divisible by 8. However, our linear reorganization in Eq. (3.14) enables omitting the basis functions with sufficiently small b_i. Concretely, the L^∞ error introduced by $R(x)$ can be controlled as $|R(x)| \le \sum_{k=1}^{4}\bar{b}_k^2 =: C_R$ for all $x \in [0,1]$, where $\{\bar{b}_k\}_{k=1}^4$ is the missed or external basis locations. Since the constant-matching construction has flexibility, $\{\bar{b}_k\}_{k=1}^4$ can be small enough to ensure $C_R \le \Delta h$. Hence, the error E'_{un} introduced by the unused

parameters satisfies

$$E'_{\text{un}} = \max_{x\in[0,1]} |\mathcal{S}'_{\text{un}}(x)| = C_R + C_c \leq 2\Delta h. \tag{3.22}$$

Here the constant C_c comes from the possible mismatch in pairing up $f_{c_t}^{\text{NN}} + f_{-c_t}^{\text{NN}}$. However, for a sufficiently wide network, C_c can be small enough as $C_c < \Delta h$.

e) Complete the proof by combining the previous estimation. By setting $\gamma = 1, \alpha = 0$ and choosing a sufficiently large $L \in \mathbb{Z}^+$, such that

$$n = \sqrt{\frac{8L^3}{\Delta h}} + 1 > \max\left\{8JL,\ \frac{8L}{\delta_s} + L,\ \frac{2L}{\Delta h}\right\}.$$

Eq. (3.21)-(3.22) gives estimation of the overall approximation error as follows:

$$\begin{aligned} \left|f_{\text{pc}}^{\text{NN}}(x) - g(x)\right| &= \left|\sum_{k\in K}\left[p_k\phi_k^+(x) + q_k\phi_k^-(x)\right] - g(x)\right| \\ &\leq |\mathcal{S}'_{\text{use}}(x) - g(x)| + |\mathcal{S}'_{\text{un}}(x)| \\ &\leq E'_{\text{use}} + E'_{\text{un}} \leq 4\Delta h, \quad \forall\, x \in [0,1], \end{aligned}$$

where Δh is chosen to satisfy $\Delta h \leq \varepsilon/8$. Hence we can finish the proof of Theorem 3.3 since $|f_{\text{pc}}^{\text{NN}}(x) - f^*(x)| \leq \varepsilon$ for all $x \in [0,1]$. □

3.3.6 Estimate the Approximation Rate

Here we estimate the approximation rate of the L^2-error e_s of approximating the single step function f_s in Eq. (3.5) by the approximator f_s^{NN} in Eq. (3.9). In our four-pair construction, we assume $s = (b_2 + b_3)/2$ and introduce k_1 and k_2 to rewrite the symmetry relations in Eq. (3.7) as:

$$\begin{aligned} b_1 &= s - k_2, \quad b_3 = s + k_1, \\ b_2 &= s - k_1, \quad b_4 = s + k_2, \end{aligned} \tag{3.23}$$

where $0 < k_1 \leq k_2$, and it also gives $d = k_2 - k_1$. The piecewise form of step function approximator f_s^{NN} in Eq. (3.9) enables us to subdivide the error into parts like

$$\begin{aligned} e_s^2 &= \int_0^1 \left| \gamma \left[f_s^{\mathrm{NN}}(x) + \frac{h}{2} \right] - \frac{h}{\Delta h} f_s(x) \right|^2 \mathrm{d}x \\ &= \gamma^2 \left[\int_{b_1}^{s} \left| f_s^{\mathrm{NN}}(x) + \frac{h}{2} \right|^2 \mathrm{d}x + \int_{s}^{b_4} \left| f_s^{\mathrm{NN}}(x) - \frac{h}{2} \right|^2 \mathrm{d}x \right]. \end{aligned} \tag{3.24}$$

After calculation, the error of each approximator f_s^{NN} can be estimated like

$$e_s^2 = \gamma^2 \left[\frac{8}{3}(k_1 - k_2)^2 \left(k_1^3 + 3k_1^2 k_2 + 2k_1 k_2^2 + k_2^3 \right) \right] \leq \frac{56}{3} \gamma^2 d^2 k_2^3. \tag{3.25}$$

During the construction of f_s^{NN}, the basis functions are chosen adjacently, leading to $k_2 \sim \mathcal{O}(d)$ and $e_s \sim \mathcal{O}(\gamma\, d^{\frac{5}{2}})$. To estimate the order of γ, we derive from Eq. (3.10) that the height h gives $h \sim \mathcal{O}(d^2)$, while the step function f_s has a d-independent height $\Delta h \sim \mathcal{O}(1)$. Therefore, the scaling $\gamma = \Delta h / h \sim \mathcal{O}(d^{-2})$ is needed, and the error is rewritten as $e_s \sim \mathcal{O}(d^{\frac{1}{2}})$. Recall that d in Eq. (3.7) has $d \sim \mathcal{O}(\frac{1}{n-1})$, we have $e_s \sim \mathcal{O}(n^{-\frac{1}{2}})$, which means the approximation rate is roughly $1/2$ order with respect to the network width n. We will numerically verify this rate in Section 4.3.

This estimation also holds for our pseudo-copy $f_{s_j,p_l}^{\mathrm{NN}}$, where $\gamma = L$. The triangle inequality is adopted to estimate the overall approximation error of these stacked $\{f_{s,p_l}^{\mathrm{NN}}\}_{l=1}^{L}$ to a predetermined step function f_s, *i.e.*,

$$e_{s,p} = \left\| \sum_{l=1}^{L} \left(f_{s,p_l}^{\mathrm{NN}} + \frac{h}{2} \right) - f_s \right\|_{L^2} \leq \sum_{l=1}^{L} \left\| \left(f_{s,p_l}^{\mathrm{NN}} + \frac{h}{2M} \right) - \frac{1}{L} f_s \right\|_{L^2} =: \sum_{l=1}^{L} e_{s,p_l}. \tag{3.26}$$

Now we focus on the approximation error e_{s,p_l} of each f_{s,p_l}^{NN} to the f_s/L. However, the result in Eq. (3.25) cannot be directly adopted since it only holds for the symmetry case in Eq. (3.23). Instead, we choose locations $\{\widetilde{b}_i\}_{i=1}^{4}$ *almost* symmetrically with

mismatch measured by Δs_l. Therefore, the relation in Eq. (3.23) is transformed into

$$\begin{aligned} \widetilde{b}_1 &= (s + \Delta s_l) - k_2, \quad \widetilde{b}_3 = (s + \Delta s_l) + k_1, \\ \widetilde{b}_2 &= (s + \Delta s_l) - k_1, \quad \widetilde{b}_4 = (s + \Delta s_l) + k_2. \end{aligned} \tag{3.27}$$

Compared with Eq. (3.23), it's clear that this transformation is equivalent to replacing $f^{\mathrm{NN}}_{s,p_l}(x)$ with $f^{\mathrm{NN}}_{s}(x - \Delta s_l)$ for all $x \in [0, 1]$. Therefore, each e_{s,p_l} in Eq. (3.26) gives

$$\begin{aligned} e^2_{s,p_l} &= \int_0^1 \left| \left[f^{\mathrm{NN}}_{s_i}(x - \Delta s_l) + \frac{h}{2} \right] - \frac{1}{L} f_s(x) \right|^2 \mathrm{d}x \\ &= \int_{b_1 + \Delta s_l}^{s} \left| f^{\mathrm{NN}}_{s_i}(x - \Delta s_l) + \frac{h}{2} \right|^2 \mathrm{d}x + \int_{s}^{b_4 + \Delta s_l} \left| f^{\mathrm{NN}}_{s_i}(x - \Delta s_l) - \frac{h}{2} \right|^2 \mathrm{d}x, \end{aligned} \tag{3.28}$$

where the integral range $[b_1 + \Delta s_l,\, b_4 + \Delta s_l]$ is not symmetrical about $x = s$ due to the mismatch. However, since Δs_l is small, we assume that $b_2 + \Delta s_l \le s \le b_3 + \Delta s_l$, then follow the similar procedure used in Eq. (3.24) to divide the error in Eq. (3.28) into parts. After calculation, we obtain the following estimation

$$\begin{aligned} e^2_{s,p_l} &= \frac{8}{3}(k_1 - k_2)^2 \left[k_1^3 + 3k_1^2 k_2 + 2k_1 k_2^2 + k_2^3 + 3\Delta s_l^2 (k_1 + k_2) \right] \\ &= \frac{8}{3} d^2 \left(-d^3 + 6d^2 k_2 - 11 d k_2^2 + 7k_2^3 - 3d\,\Delta s_l^2 + 6k_2\,\Delta s_l^2 \right). \end{aligned}$$

Since Δs_l can be assumed to be small as $\Delta s_l \sim \mathcal{O}(d)$, we obtain a similar estimation $e_{s,p_l} \sim \mathcal{O}(d^{\frac{5}{2}})$. Since the number of stacked pseudo-copy satisfies $L = \gamma \sim \mathcal{O}(d^{-2})$, the same estimation $e_{s,p} = L e_{s,p_l} \sim \mathcal{O}(n^{-\frac{1}{2}})$ is achieved.

3.3.7 Proof of Theorem 3.4

Extending our results of permutation-trained networks to the random initialization scenarios imposes non-trivial challenges. Note that the symmetry construction of the step function approximator in Eq. (3.6) becomes invalid with random initializations, as the error introduced by randomness can accumulate as the width

increases. Nevertheless, the randomly sampled parameters will become denser upon increasing width, leading to a high probability of finding parameters that closely match the required location.

Therefore, we can first apply the UAP in the equidistant case to obtain a network f^{NN} in Eq. (3.2) (denoted as $f^{\mathrm{NN}}_{\mathrm{equi}}$ in the following text for distinction), which exhibits approximation power. Then, within a randomly initialized network f^{NN} in Eq. (3.2) (denoted as $f^{\mathrm{NN}}_{\mathrm{rand}}$) of sufficient width, we find a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ that can be regarded as randomly perturbed from $f^{\mathrm{NN}}_{\mathrm{equi}}$. If this perturbation is small enough, the subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ will also possess approximation power.

Proof of Theorem 3.4. We divide the whole proof into the following points:

a) Approach f^* with equidistantly initialized $f^{\mathrm{NN}}_{\mathrm{equi}}$. Theorem 3.2 indicate that for any small ε and the target function f^*, there is an equidistantly initialized network $f^{\mathrm{NN}}_{\mathrm{equi}}$ in Eq. (3.2) with width $\widetilde{n}$ and $\mathbf{b}^{(\widetilde{n})}_{\mathrm{equi}} = (b_i)_{i=1}^{\widetilde{n}}$, $\mathbf{w}^{(2\widetilde{n})}_{\mathrm{equi}} = (\pm b_i)_{i=1}^{\widetilde{n}}$, such that

$$\left|f^{\mathrm{NN}}_{\mathrm{equi}}(x) - f^*(x)\right| < \frac{\varepsilon}{4}, \quad \forall\, x \in [0,1]. \tag{3.29}$$

b) Find a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ in a randomly initialized $f^{\mathrm{NN}}_{\mathrm{rand}}$ to approximate $f^{\mathrm{NN}}_{\mathrm{equi}}$. For a network with sufficiently wide $n \gg \widetilde{n}$, parameters $\mathbf{b}^{(n)}_{\mathrm{rand}} \sim \mathcal{U}[0,1]^n$ and $\mathbf{w}^{(2n)}_{\mathrm{rand}} = (\pm p_i)_{i=1}^{n}$, $p_i \sim \mathcal{U}[0,1]$, we can find a subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ with parameters

$$\mathbf{b}^{(\widetilde{n})}_{\mathrm{sub}} = \left(b_i + r_i^{\mathbf{b}}\right)_{i=1}^{\widetilde{n}}, \quad \mathbf{w}^{(2\widetilde{n})}_{\mathrm{sub}} = \left(\pm\left(b_i + r_i^{\mathbf{w}}\right)\right)_{i=1}^{2\widetilde{n}},$$

which can be viewed as randomly perturbed from $\mathbf{b}^{(\widetilde{n})}_{\mathrm{equi}}$, $\mathbf{w}^{(2\widetilde{n})}_{\mathrm{equi}}$, while the independent and identically distributed (i.i.d.) $r_i^{\mathbf{b}} \sim \mathcal{U}[-\Delta r, \Delta r]^{\widetilde{n}}$ and $r_i^{\mathbf{w}} \sim \mathcal{U}[-\Delta r, \Delta r]^{2\widetilde{n}}$ measured the perturbation, and $\Delta r > 0$ are the maximum allowable perturbation of $\mathbf{b}^{(\widetilde{n})}_{\mathrm{equi}}$ and $\mathbf{w}^{(2\widetilde{n})}_{\mathrm{equi}}$. (We impose further constraints on the parameters near the boundary of $[0,1]$ to prevent them from exceeding the range.) Consequently, for sufficiently small $\Delta r < r_0$, the subnetwork $f^{\mathrm{NN}}_{\mathrm{sub}}$ will approach $f^{\mathrm{NN}}_{\mathrm{equi}}$ with the approximation error

like

$$\left|f_{\text{sub}}^{\text{NN}}(x) - f_{\text{equi}}^{\text{NN}}(x)\right| < \frac{\varepsilon}{4}, \quad \forall\, x \in [0,1], \qquad \text{with probability } P_{\text{sub}}, \tag{3.30}$$

where P_{sub} denote the related possibility. This also enables $f_{\text{sub}}^{\text{NN}}$ the approximation power to f^* due to its continuity with respect to the parameters. Combining the results in Eq. (3.29)-(3.30), the approximation error $E_{\text{rand}}^{\text{sub}}$ of the subnetwork gives that, with probability P_{sub},

$$\begin{aligned} E_{\text{rand}}^{\text{sub}} &:= \max_{x\in[0,1]} \left|f_{\text{sub}}^{\text{NN}}(x) - f^*(x)\right| \\ &\leq \max_{x\in[0,1]} \left|f_{\text{sub}}^{\text{NN}}(x) - f_{\text{equi}}^{\text{NN}}(x)\right| + \max_{x\in[0,1]} \left|f_{\text{equi}}^{\text{NN}}(x) - f^*(x)\right| < \frac{\varepsilon}{2}. \end{aligned} \tag{3.31}$$

c) Estimate the probability P_{sub} related to the approximation power of $f_{\text{sub}}^{\text{NN}}$. Here we estimate the probability of finding such a subnetwork $f_{\text{sub}}^{\text{NN}}$ with the required approximation error $E_{\text{rand}}^{\text{sub}}$ in Eq. (3.31). We start with the complement event $\mathcal{A}_k^c$: for a given location $\hat{b}_k \in \mathbf{b}_{\text{equi}}^{(\widetilde{n})}$, there is no close enough locations in $\mathbf{b}_{\text{rand}}^{(n)}$, *i.e.*, falls in the interval $[\hat{b}_k - \Delta r, \hat{b}_k + \Delta r]$. Concretely, the probability has $\mathbb{P}[\mathcal{A}_k^c] = (1-2\Delta r)^n$ since the interval length is $2\Delta r$. Hence, by choosing Δr small enough such that these intervals have no overlap, *i.e.*, $\Delta r < \min\{\delta_s, 1/2\widetilde{n}\}$, we apply the inclusion-exclusion principle [25] to write the probability of finding all locations $\mathbf{b}_{\text{equi}}^{(\widetilde{n})}$ in $\mathbf{b}_{\text{rand}}^{(n)}$ as

$$\mathbb{P}\left[\bigcap_{k=1}^{\widetilde{n}} \mathcal{A}_k\right] = 1 - \mathbb{P}\left[\bigcup_{k=1}^{\widetilde{n}} \mathcal{A}_k^c\right] = 1 - \sum_{k=1}^{\widetilde{n}} (-1)^{k+1} \binom{\widetilde{n}}{k} (1-2k\Delta r)^n =: 1 - P',$$

where $\binom{\widetilde{n}}{k}$ is the binomial coefficient, and P' is the complement probability. This probability also holds for finding $\mathbf{w}_{\text{equi}}^{(2\widetilde{n})}$ in pairwise $\mathbf{w}_{\text{rand}}^{(2n)}$. Consequently, we can find a subnetwork $f_{\text{sub}}^{\text{NN}}$ within $f_{\text{rand}}^{\text{NN}}$ that is close enough to $f_{\text{equi}}^{\text{NN}}$, such that the approximation error $E_{\text{rand}}^{\text{sub}}$ in Eq. (3.31) is achieved with the probability of

$$P_{\text{sub}} = \mathbb{P}\left[E_{\text{rand}}^{\text{sub}} < \frac{\varepsilon}{2}\right] = \left[1 - P'\right]^2. \tag{3.32}$$

For given $\widetilde{n}$ and Δr, we have $P' \to 0$ as $n \to \infty$. Therefore, we choose a sufficiently large n to ensure that the probability $P_{\text{sub}} \geq \sqrt{1-\delta}$.

d) Annihilate the remaining part in the randomly initialized $f_{\text{rand}}^{\text{NN}}$. We follow the same discussion as in the equidistant case to eliminate the unused parameters in $f_{\text{rand}}^{\text{NN}}$. By applying linear reorganization and Lemma 3.7, we rewrite the remaining part as a linear function $\mathcal{S}_{\text{un}}^{r}(x) = \beta_r x + \eta_r$, where $x \in [0,1]$ and $0 \leq \beta_r \leq \Delta p$. The upper bound Δp is the largest gap between the adjacent coefficients $\{p_i\}_{i\in I_{\text{un}}}$. As the network width n increases, the randomly initialized $\{p_i\}_{i\in I_{\text{un}}}$ become denser, leading to $\Delta p \xrightarrow{\text{a.s.}} 0$. Therefore, we can choose a sufficiently large n to ensure $\Delta p \leq \varepsilon/2$ with high probability, *i.e.*, $P_{\text{un}} \geq \sqrt{1-\delta}$, where P_{un} denotes the related possibility.

Similarly to Eq. (3.19), we introduce an additional shift $C_r = -\eta_r$ to control the error of the intercept η_r. Therefore, the error $E_{\text{rand}}^{\text{un}}$ introduced by the unused part in $f_{\text{rand}}^{\text{NN}}$ satisfies can be estimated similarly with Eq. (3.19) as the following form,

$$E_{\text{rand}}^{\text{un}} = \max_{x\in[0,1]} |\mathcal{S}_{\text{un}}^{r}(x) + C_r| \leq \Delta p \leq \frac{\varepsilon}{2}, \quad \text{with probability } P_{\text{un}} > \sqrt{1-\delta}. \tag{3.33}$$

e) Complete the proof by combining the previous estimation. For any $\varepsilon > 0$ and $f^* \in C([0,1])$, we choose $\Delta r < \min\{r_0, \delta_s, 1/2\widetilde{n}\}$ and a large n to ensure that

1. With probability P_{sub}, the subnetwork $f_{\text{sub}}^{\text{NN}}$ can approximate the f^* with the error $E_{\text{rand}}^{\text{sub}}$ satisfies $E_{\text{rand}}^{\text{sub}} < \varepsilon/2$ based on Eq. (3.31);
2. The probability P_{sub} satisfies $P_{\text{sub}} \geq \sqrt{1-\delta}$ based on Eq. (3.32);
3. With probability $P_{\text{un}} > \sqrt{1-\delta}$, the impact of the remaining parameters satisfies $E_{\text{rand}}^{\text{un}} < \varepsilon/2$ based on Eq. (3.33).

Hence, we can finish the proof by estimating the overall approximation error of our network $f_{\text{rand}}^{\text{NN}}$ to the target function f^*, which gives that with probability $1-\delta$,

$$\left|f_{\text{rand}}^{\text{NN}}(x) - f^*(x)\right| \leq E_{\text{rand}}^{\text{sub}} + E_{\text{rand}}^{\text{un}} < \varepsilon, \quad \forall\, x \in [0,1]. \tag{3.34}$$

□

3.4 Experiments

This section presents numerical evidence to support and validate the theoretical proof. An interesting observation of permutation behaviors also highlights the theoretical potential of this method. The code and data are publicly available on GitHub at https://github.com/DanclaChen/PermutationTraining.

3.4.1 Algorithmic Implementation

To find a reference for the permutation, the *lookahead permutation (LaPerm)* algorithm introduced a k-times Adam-based free updating, where the learned relationship can then serve as a reference for permutation [71]. The impact of k's value on convergence behavior is evaluated to be negligible (see Section 3.4.6). Apart from the fixed permutation period k, it's also possible to adjust k to learn sufficient information for the next permutation.

3.4.2 Experimental Settings

To validate our theoretical results, we conduct experiments on regression problems. The settings are deliberately chosen to be consistent with our proof construction. We consider a three-layer network in Eq. (3.2), where the first hidden layer's parameters are fixed to form the basis functions $\{\phi_i^{\pm}\}_{i=1}^n$ in Eq. (3.1). The weights $\boldsymbol{\theta}^{(2n)}$ of the second hidden layer are trained by permutation, while the scaling factors α, γ in the output layer are freely trained to reduce the required network width.

To establish the convergence property upon increasing network width, we sample the training points randomly and uniformly in $[-1, 1]$, along with equidistantly distributed test points. The maximum training epoch is sufficiently large to ensure reaching the stable state. For the multi-dimensional case, we set the basis functions at a larger domain than the functions to ensure accuracy near the boundary. The scale is measured by T_b, which means the biases are in $[-1 - T_b, 1 + T_b]$ in each dimension. See Table 3.1 for detailed choice.

Table 3.1: Hyperparameter setting of permutation trained NNs.

Hyperparameters	1D	2D	3D
Architectures	1-2n-1-1	2-8n-1-1	3-26n-1-1
k	5	5	20
Batch size	8	128	640
# training points	1600	51200	
# test points	400	12800	
T_b	0	0.75	
n	$\{10, 20, 40, 80, 160, 320\}$		
# epoch	6400		
Learning rate (LR)	1e-3		
Multiplicative factor of LR decay	0.998		
Multiplicative factor of k increase	$\sqrt[10]{1.002}$		

The experiments are conducted on an NVIDIA A100 GPU. However, our code is hardware-friendly since each case only consumes approximately 2GB of memory. The code is implemented in the `PyTorch` library [70]. The error bars mark the range of the maximum and minimum values with ten different random seeds. Additionally, the training data of each case is sampled under the random seed 2022 to ensure that they are comparable.

3.4.3 Approximating the One-Dimensional Continuous Functions

We utilize a 1-2n-1-1 network architecture with equidistant initialization discussed in Theorem 3.2, pairwise random initializations in Theorem 3.4, and also totally random initialization $\mathbf{w}^{(2n)} \sim \mathcal{U}[-1,1]^{2n}$. The approximation targets are sine function $y = -\sin(2\pi x)$ and 3-order Legendre polynomial $y = \frac{1}{2}(5x^3 - 3x)$, $x \in [-1,1]$.

The numerical result illustrated in Figure 3.3 exhibits a clear convergence behavior of equidistant and pairwise random cases upon increasing n, agreeing with our theoretical proof. The clear dominance of pairwise random initialization indicates its undiscovered advantages in permutation training scenarios. Besides, the total random case also shows a certain degree of approximation power. However, to attain the equivalent accuracy, the total random case requires a wider network (*i.e.*, a larger n). Furthermore, the L^∞ error exhibits a $1/2$ convergence rate with respect to n. Although the theoretical estimation in Section 3.3 is based on L^2 norm, we indeed observe that it also holds for L^∞ error.

3.4.4 Approximating the Multi-Dimensional Continuous Functions

As a natural extension, we consider a two-dimensional function $z = -\sin \pi xy$, where $(x,y) \in [-1,1]^2$, starting with the construction of basis functions like $\phi_i^{\pm}(x)$

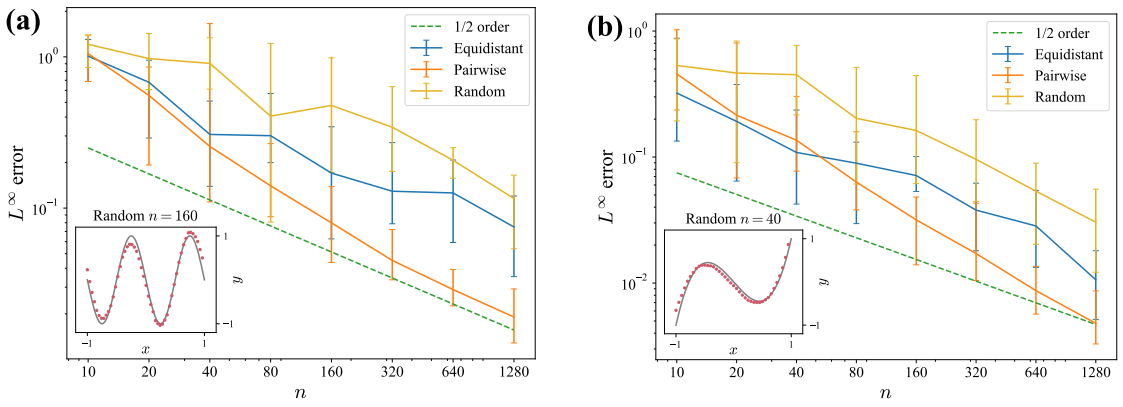

Figure 3.3: Approximating one-dimensional continuous function (a): $y = -\sin(2\pi x)$ and (b): $y = \frac{1}{2}(5x^3 - 3x)$ with equidistantly, pairwise random, and randomly initialized network, where $x \in [-1, 1]$. The inset in each panel presents the target function as lines and an example of the approximation result as dots.

in Eq. (3.1). Recall that in the one-dimensional case, the two subsets of basis $\phi_i^{\pm}(x)$ correspond to the two opposite directions along the x-axis. Therefore, at least four directions are required to represent a function defined on the xy-plane, of which two are parallel to the x-axis as $\phi_i^{\pm}(x, \cdot) = \mathrm{ReLU}(\pm(x - b_i))$ and $\phi_j^{\pm}(\cdot, y) = \mathrm{ReLU}(\pm(y - b_j))$ for y-axis, respectively. Furthermore, inspired by the lattice Boltzmann method in fluid mechanics [12], we introduce another four directions as $\psi_k^{\pm\pm}(x, y) = \mathrm{ReLU}(\pm x \pm y - b_k)$. So the whole basis functions are divided into eight subsets, each corresponding to a different direction (see Figure 3.4(b)). Also, the range of biases is essential since the distribution of $\psi_k^{\pm\pm}(x, y)$ must be at least $\sqrt{2}$-times wider to cover the entire domain. Here, we set the biases to range in varying directions with a uniform scaling factor, providing flexibility in dealing with the unwanted coefficients.

Accordingly, we utilize a 2-8n-1-1 network architecture and follow the same setting as before (refer to Table 3.1). The results depicted in Figure 3.4(a) also show good approximation power. However, the 1/2 convergence rate in previous cases cannot be attained here. We hypothesize that this is due to our preliminary eight-direction setting of the basis functions. This degeneration indicates the challenge of

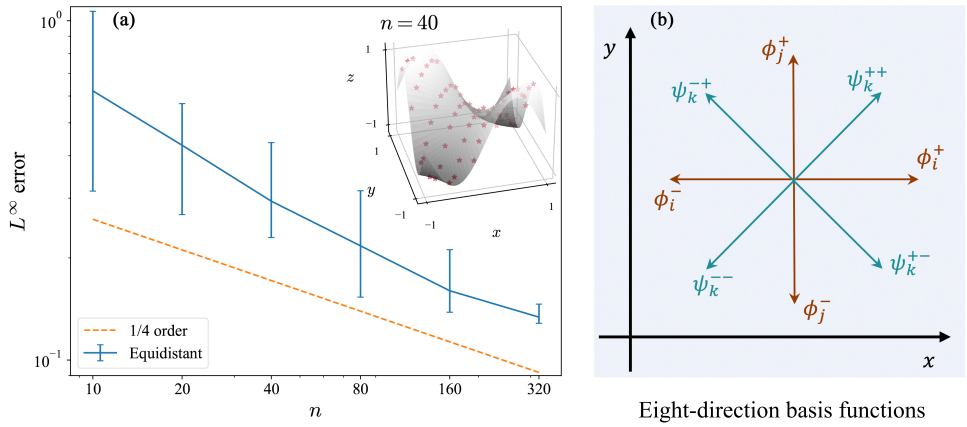

Figure 3.4: (a) Approximating two-dimensional continuous function $z = -\sin \pi xy$, where $x, y \in [-1, 1] \times [-1, 1]$. The inset panel presents the target function surface and an example of the approximation result as dots. (b) The two-dimensional basis function settings.

extending our theoretical results to higher dimensions. Further research will address this difficulty by considering more appropriate high-dimensional basis function setups. One possible approach relies on the basis construction utilized in the finite element methods [4]. However, adopting such a method to meet the permutation training scenario is non-trivial and requires further investigation.

Moreover, the mismatch between the existing implementations and the permutation setting poses numerical challenges to permutation training in higher dimensions. The performances are significantly affected by the algorithm implementations and initialization settings, both of which need further investigation and are beyond the scope of this work. We hope our work can inspire and motivate the development of more sophisticated implementations specific to permutation training. However, the permutation training, as a numerical algorithm, can be directly applied to high-dimensional cases, even if it requires a significantly larger network width.

Using a similar numerical setting, we can also approximate functions with three-dimensional inputs. Here we consider $f(x, y, z) = \sin 3x \cdot \cos y \cdot \sin 2z$, where

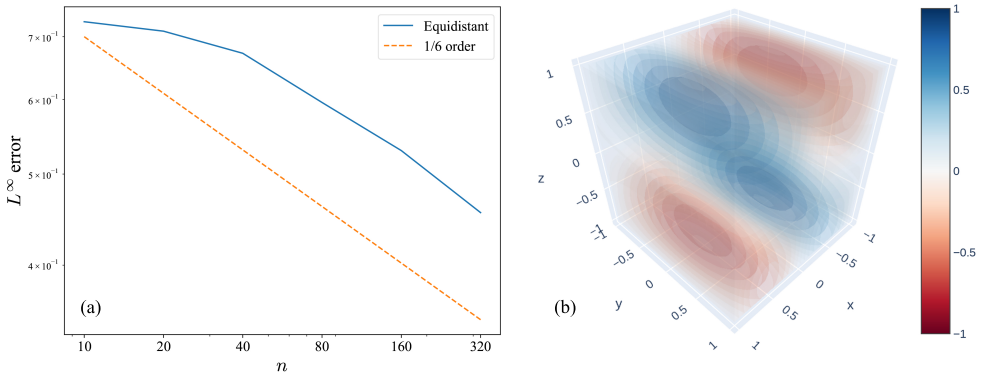

Figure 3.5: Approximating three-dimensional continuous function $f(x, y, z) = \sin 3x \cdot \cos y \cdot \sin 2z$, where $(x, y, z) \in [-1, 1]^3$. (a) The convergence behavior under random seed 2022. (b) The three-dimensional illustration of the target function, where the function value $f(x, y, z)$ is plotted by the corresponding color in the color bar.

$(x, y, z) \in [-1, 1]^3$. The results plotted in Figure 3.5 demonstrate a certain degree of approximation power (due to the computational cost's limitation, we only conduct the experiments once). However, a degeneration convergence rate from $1/2$ to $1/6$ also indicates the theoretical limitations of the current construction.

3.4.5 Permutation-Trained NNs with Leaky-ReLU

In addition to the ReLU activation function, we also numerically consider the leaky-ReLU activation function $\sigma(x) = \max(0, x) + \alpha \min(0, x)$, where α is a small constant. The leaky-ReLU is widely used since it can alleviate the dying ReLU problem by allowing a small gradient when the unit is not active.

Extending our UAP results to leaky-ReLU is expected. This is because of the two crucial techniques deployed in our proof: constructing the step function approximators and eliminating the unused parameters, both can be applied to the leaky-ReLU. Since our two-direction setting of basis function $\phi^{\pm}$ in Eq. (3.1) can impart leaky-ReLU with symmetry equivalent to ReLU, it's feasible to construct a similar step function approximator by re-deriving the relevant coefficients p_i, q_i.

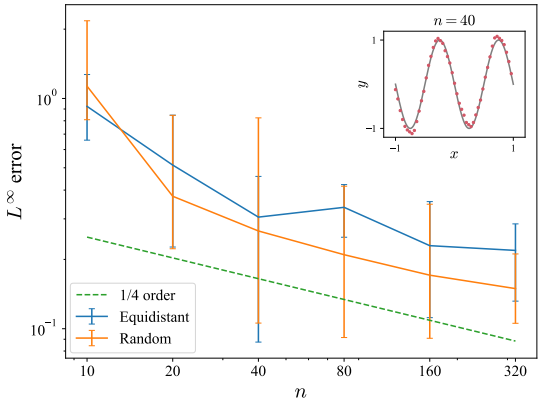

Figure 3.6: Approximating one-dimensional continuous function $y = -\sin(2\pi x)$ with equidistantly initialized network equipped with leaky-ReLU, where $x \in [-1, 1]$. The inset in each panel presents the target function as lines and an example of the approximation result as dots.

Furthermore, the existing elimination method can be directly employed since a pair of leaky-ReLU basis functions can be constructed into linear functions for further processing.

As an initial attempt, we numerically examine the leaky-ReLU networks by only changing the activation function in the case of Figure 3.3(a). The results plotted in Figure 3.6 exhibit the approximation power of leaky-ReLU networks. Unlike the ReLU cases, the random initialization outperforms the equidistant initialization. However, the 1/2 convergence rate in previous ReLU cases cannot be attained here, probably because the proof based on leaky-ReLU may result in different constants, leading to potential discrepancies in details when compared with the ReLU-based conclusions.

3.4.6 The Impact of Permutation Period

As a hyperparameter, the choice of permutation period k during the implementation of LaPerm algorithms may affect the convergence behavior. The correlation between the value of k and the final accuracy is reported to be unambiguous (refer

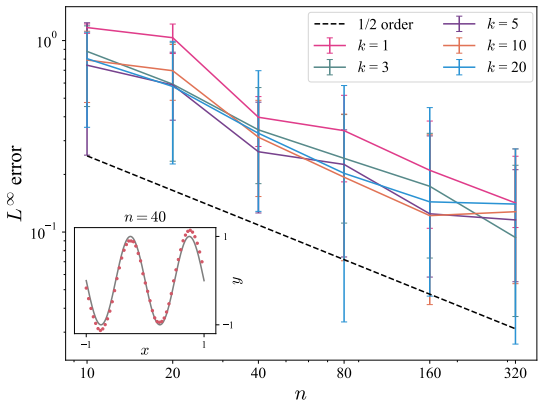

Figure 3.7: Approximating one-dimensional continuous function $y = a_0 + a_1 \sin(\pi x) + a_2 \cos(2\pi x) + a_3 \sin(3\pi x)$ with equidistantly initialized network, where $x \in [-1, 1]$, and the value of permutation period $k = 1, 3, 5, 10, 20$, respectively. The inset in each panel presents the target function as lines and an example of the approximation result as dots.

to Figure 6 in [71]). Generally, a larger k is associated with slightly higher accuracy of single permutation training result, thus, in our experiments, the weights are permuted after each k epoch. Figure 3.7 evaluates the impact of k's value on convergence behavior, whose results suggest that this effect remains negligible.

3.4.7 The Influence of Initialization Strategies

Here, we explore the effect of different initialization choices on the approximation behavior, which holds greater significance in permutation training scenarios due to the preservation of the initialized values. The case in Figure 3.3(a) is utilized to apply various random initialization strategies. The results plotted in Figure 3.8 show that the UAP of permutation-trained networks is not limited to the setting considered by our previous theoretical investigation. For more generalized cases, we first consider randomly initializing only $\mathbf{w}^{(2n)}$ and $\mathbf{b}^{(n)}$, which are labeled as Random 3 and 4, respectively. Both cases demonstrate competitive or even superior accuracy.

Next, we consider some commonly used initializations, such as Xavier's uniform

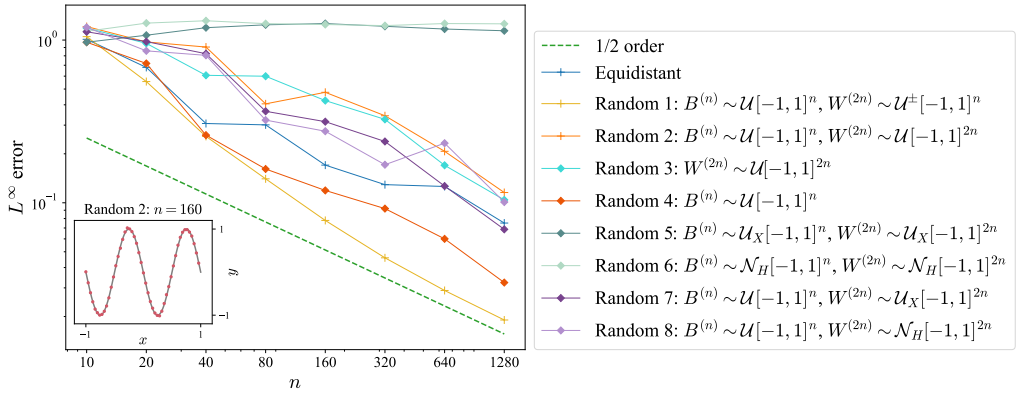

Figure 3.8: The performance of different initialization strategies in approximating $y = -\sin(2\pi x)$ in $[-1, 1]$. The pairwise initialization $\mathbf{w}^{(2n)} = (\pm p_i)_{i=1}^n$, $p_i \sim \mathcal{U}[-1, 1]$ is denoted as $\mathbf{w}^{(2n)} \sim \mathcal{U}^{\pm}[0, 1]^n$. The error bars are omitted for conciseness. The inset panel presents the target function as lines and an example of the approximation result as dots.

initialization U_X [31], and He's normal initialization N_H [36]. However, the implementation of U_X on Random 5 and N_H on Random 6 fails to result in convergence. This abnormal poor performance may be attributed to the mismatch of the scale in $\mathbf{b}^{(n)}$ and the approximation domain $[0, 1]$. To verify our hypothesis, we hold $\mathbf{b}^{(n)} \sim \mathcal{U}[-1, 1]^n$, and only apply U_X and N_H on the coefficients $\mathbf{w}^{(2n)}$ in Random 7 and 8. Consequently, both cases successfully regain the approximation ability. These surprising behaviors, especially the unexpected deterioration of the default choices, emphasizes the limited understanding of systematic characterization of the initialization suitable for permutation training scenarios.

3.4.8 Observation of Permutation Patterns

This section aims to explore the theoretical potential of permutation training in describing network learning behavior. Based on the significant correlation between permutation and learning behavior, as evidenced by [71] and our investigation, we hypothesize that the permutation-active components of the weights may play a crucial role in the training process. Therefore, by identifying and tracing the

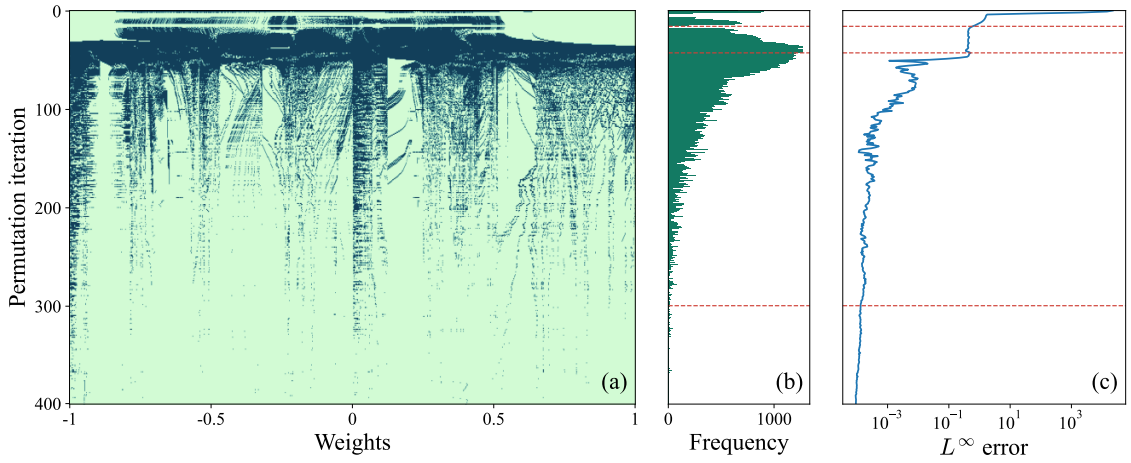

Figure 3.9: The permutation behavior in the first 400 permutation iterations in approximating $y = -\sin(2\pi x)$ by an equidistantly initialized network with $n = 640$. (a) The distribution of the active components (denoted by dark green color). (b) The frequency distribution illustrates the variation in the total count of active components in each permutation. (c) The corresponding loss behavior.

permutation-active part of weights, a novel description tool can be achieved, which also facilitates visualization and statistical analysis of the learning behavior.

As a preliminary attempt, we illustrate the permutation behavior of the coefficients $\boldsymbol{\theta}^{(2n)}$ in Figure 3.3(a). The components that participated in the permutation are visually highlighted in dark green in Figure 3.9(a). The behaviors plotted in Figure 3.9(b)-(c) clearly show that the frequency of order relationship exchange evolves synchronously with the learning process, agreeing with the observation of [71].

Specifically, the distribution of active components shows significant patterns, which are classified into four stages (marked by red dashed lines in Figure 3.9). The loss declines sharply in the initial stage, while only the components with medium value are permuted. Once loss reaches a plateau in the second stage, more components are involved in permutation, evidencing the role of permutation in propelling the training. As loss starts to decline again, the permutation frequency correspondingly diminishes. Interestingly, the slower loss decrease gives rise to a ribbon-like pattern, akin to the reported localized permutations (plotted in Figure 14 of the

appendix in [71]), and possibly due to slow updates failing to trigger a permutation. This observation may support the existence of inherent low-dimensional structures within the permutation training dynamics, potentially linked to mathematical depiction of permutation groups, such as cycle decomposition [8] and Fourier bases for permutation [41]. Finally, the permutation's saturation aligns with the stationary state of loss convergence. We believe these inspiring phenomena deserve further exploration, as they hold the potential to provide novel insights regarding the learning behavior of networks.

Chapter 4

Sidecar: A Structure-Preserving Framework of Solving PDEs with NNs

In this chapter, we introduce *Sidecar*, an innovative framework aimed at improving function-learning NN solvers for temporal evolution PDEs by embedding structure-preserving knowledge. Drawing inspiration from the TDSR method discussed in Section 2.6.2, Sidecar incorporates a lightweight, time-dependent copilot NN to model the evolution of preserved quantities. This copilot network guides the primary NN solver to adhere to structure-preserving properties. Experimental results demonstrate that Sidecar significantly enhances solution accuracy and physical consistency for both conservative and dissipative PDEs. Additionally, an ablation study highlights the framework's robustness and the effectiveness of its design [11]. Our main findings are summarized below:

1. We introduce Sidecar, a framework enhancing NN solvers for PDEs by embedding structure-preserving knowledge via a lightweight copilot NN.

2. Sidecar's adaptability is demonstrated to various NN solvers and PDEs, improving physical consistency without performance loss.

3. We numerically validate Sidecar's effectiveness on benchmark PDEs, achieving higher accuracy and consistency, supported by ablation studies.

4.1 Introduction

This section discusses the related works in the field of structure-preserving numerical methods and NN solvers for PDEs and outlines of this chapter. It serves as a complement to the introduction in Section 1.2.

4.1.1 Related Works

Recent works have attempted to incorporate the structure-preserving knowledge into NN solvers [30,37,42,51], but these approaches often impose additional training challenges, and thus sacrifice the performance and computational efficiency. Existing structure-preserving methods for NN solvers can generally be divided into two categories: **a)** *hard constraints*: to manually post-process or project the network's outputs to enforce the physical structure [30, 37], and **b)** *soft regularization*: to introduce additional regularization terms into the loss function [42, 51]. Ideally, the structure-preserving properties should facilitate the learning process of NN solvers rather than impose constraints.

However, a common challenge of these methods is the undesired trade-off between accuracy and physical fidelity, ending up with a performance degradation. Additionally, *hard constraints* methods often suffer from distribution shifts between training and testing data, which can hurt the generalization ability. On the other hand, the commonly-used *soft regularization* methods may face the challenge of numerical integration, as the preserved quantities often involve integration over the spatial domain. The numerical integration algorithms are required to be differentiable for back-propagation, which can be impractical in scenarios with discontinuities or singularities.

4.1.2 Outlines

The remainder of this chapter is structured as follows: Section 4.2 describes the proposed Sidecar framework, including the loss function design and training

procedure. Section 4.3 evaluates Sidecar's performance on benchmark PDEs and compares it with existing NN solvers. Section 4.4 presents the ablation study and further discussions on the advantages of the proposed framework.

4.2 Methodology

This section presents a novel structure-preserving framework named *Sidecar* to improve the physical consistency of the existing function-learning NN solvers. Here we discuss its architecture, loss function, and training strategy.

4.2.1 Framework Architecture

Inspired by the TDSR method discussed in Section 2.6.2, which introduces a time-dependent factor to incorporate the structure-preserving properties Eq. (2.8) into the PDEs (2.4), we also rewrite the numerical solution as

$$\bar{u}_{\mathrm{NN}}(\mathbf{x}, t) = \bar{R}_{\mathrm{NN}}(t)\, \bar{v}_{\mathrm{NN}}(\mathbf{x}, t), \tag{4.1}$$

where $\bar{v}_{\mathrm{NN}}(\mathbf{x}, t)$ is the primary network to learn the PDE solution, and $\bar{R}_{\mathrm{NN}}(t)$ is the time-dependent copilot network to guide $\bar{v}_{\mathrm{NN}}(\mathbf{x}, t)$ to respect the structure-preserving properties. The overall solution $\bar{u}_{\mathrm{NN}}(\mathbf{x}, t)$ satisfies the coupled system in Eq. (2.18). The Sidecar framework is illustrated in Figure 4.1, where the *primary-copilot* design allows Sidecar to be flexibly integrated with existing NN solvers.

As for the architecture choice of each part, the primary network $\bar{v}_{\mathrm{NN}}(\mathbf{x}, t)$ can inherit the architecture of existing NN solvers, such as the MLP in Eq. (2.1) adopted in the vanilla PINNs [72]. Meanwhile, to maintain computational efficiency and avoid overwhelming $\bar{v}_{\mathrm{NN}}(\mathbf{x}, t)$, the copilot network $\bar{R}_{\mathrm{NN}}(t)$ is implemented to be lightweight, such as a shallow MLP with significantly fewer neurons compared to $\bar{v}_{\mathrm{NN}}(\mathbf{x}, t)$.

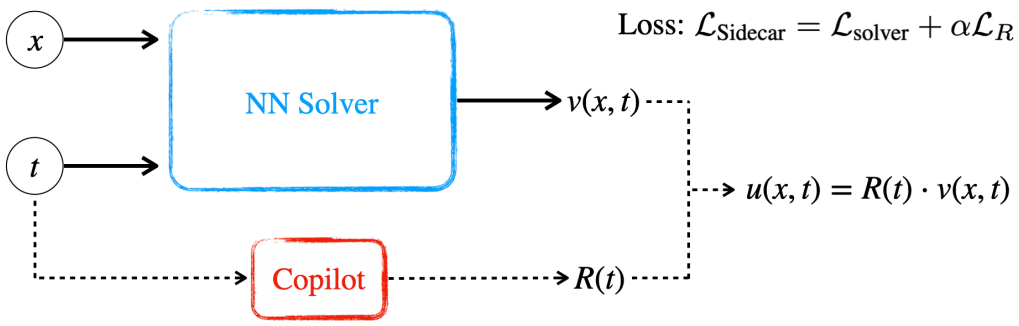

Figure 4.1: The architecture of the Sidecar framework.

4.2.2 Loss Function

Incorporating structure-preserving properties into network training requires a well-designed loss function, especially when the structure ODE (2.19) contains spatial integration operators. Here, we design the Sidecar loss to consist of two components:

$$\mathcal{L}_{\text{Sidecar}}[\bar{R}_{\text{NN}}, \bar{v}_{\text{NN}}] = \mathcal{L}_{\text{solver}}[\bar{R}_{\text{NN}}\, \bar{v}_{\text{NN}}] + \alpha\mathcal{L}_R[\bar{R}_{\text{NN}}, \bar{v}_{\text{copy}}], \tag{4.2}$$

where $\mathcal{L}_{\text{solver}}$ is the solver loss that guides the solution to learn the PDE solution, $\mathcal{L}_R$ is the structure loss that ensures the copilot network $\bar{R}_{\text{NN}}(t)$ effectively guides the primary network $\bar{v}_{\text{NN}}(\mathbf{x}, t)$ to adhere to the physical properties, α is a hyperparameter to control the relative weight, and $\bar{v}_{\text{copy}}$ is a detached copy of the primary network $\bar{v}_{\text{NN}}$ (and will be discussed in detail in Section 4.2.2). While $\mathcal{L}_{\text{Sidecar}}$ adopts the typical *main-regularization* format, its implementation is specifically tailored for Sidecar.

Remark 4.1. Our experiments set $\alpha = 1$ as the default value. While fine-tuning α can lead to further improvements, the advantages of Sidecar remain robust to the choice of α.

The Solver Loss

We design $\mathcal{L}_{\text{solver}}[\bar{R}_{\text{NN}}\, \bar{v}_{\text{NN}}]$ to evaluate the combined output rather than primary network $\bar{v}_{\text{NN}}(\mathbf{x}, t)$ only, allowing the copilot network $\bar{R}_{\text{NN}}$ to learn PDE-related information. The design of $\mathcal{L}_{\text{solver}}$ can be inherited from the chosen NN solver. In the case of vanilla PINNs [72], $\mathcal{L}_{\text{solver}}$ can adopt the PINNs loss function as defined in Eq. (2.5):

$$\mathcal{L}_{\text{solver}}[\bar{R}_{\text{NN}}, \bar{v}_{\text{NN}}] = \mathcal{L}_{\text{PINNs}}[\bar{R}_{\text{NN}}\, \bar{v}_{\text{NN}}]. \tag{4.3}$$

The Structure Loss

A direct but impractical approach is to define the structure loss $\mathcal{L}_R$ as the residual of the structure ODE (2.19), where spatial integration operators $\mathcal{I}_{\mathcal{Q}}[\bar{v}_{\text{NN}}]$ and $\mathcal{I}_{\mathcal{S}}[\bar{v}_{\text{NN}}]$ are evaluated using numerical integration algorithms based on $\bar{v}_{\text{NN}}(\mathbf{x}, t)$. However, this approach requires the numerical integration algorithms to be differentiable for back-propagation. While several differentiable numerical integration algorithms exist, such as `torchquad` [32] or `torch.trapezoid` in `PyTorch` library [70], they are mainly based on low-order numerical schemes and are invalid for complex PDE systems with discontinuities or singularities.

Numerical integration implementation: To facilitate the incorporation of the existing high-accuracy numerical integration schemes, we propose to compute the integration $\mathcal{I}_{\mathcal{Q}}$ and $\mathcal{I}_{\mathcal{S}}$ based on $\bar{v}_{\text{copy}}(\mathbf{x}, t)$, a detached copy of $\bar{v}_{\text{NN}}(\mathbf{x}, t)$. The adopted numerical integration algorithm is the Romberg integration [29], which is a widely used scheme with high accuracy. Subsequently, we minimize the structure loss $\mathcal{L}_R$ with respect to $\bar{R}_{\text{NN}}(t)$ only, while keeping $\bar{v}_{\text{NN}}(\mathbf{x}, t)$ detached from the back-propagation process, *i.e.*, $\min_{\bar{R}_{\text{NN}}} \mathcal{L}_R[\bar{R}_{\text{NN}}, \bar{v}_{\text{copy}}]$.

Temporal discretization: To avoid back-propagating through the numerical integration algorithms, we need to discretize the structure ODE (2.19) into N_T time points. For simplicity, we consider a regular grid for the time points $t^n = n\,\delta t$,

$n = 0, 1, \cdots, N_T$, where $\delta t = T/N_T$ and denote discrete variables and operators as

$$R^n := R(t^n), \qquad v^n(\mathbf{x}) := v(\mathbf{x}, t^n), \qquad \hat{\mathcal{I}}_{\mathcal{Q}}[v^n] = \int_\Omega \mathcal{K}^v_{\mathcal{Q}}[v^n](\mathbf{x})\,\mathrm{d}\mathbf{x},$$
$$\bar{R}^n_{\mathrm{NN}} := \bar{R}_{\mathrm{NN}}(t^n), \quad \bar{v}^n_{\mathrm{copy}}(\mathbf{x}) := \bar{v}_{\mathrm{NN}}(\mathbf{x}, t^n), \quad \hat{\mathcal{I}}_{\mathcal{S}}[v^n] = \int_\Omega \mathcal{K}^v_{\mathcal{S}}[v^n](\mathbf{x})\,\mathrm{d}\mathbf{x},$$

where $\hat{\mathcal{I}}_{\mathcal{Q}}, \hat{\mathcal{I}}_{\mathcal{S}} : (\Omega \to \mathbb{R}) \to \mathbb{R}$ are the discrete version of the integration operators $\mathcal{I}_{\mathcal{Q}}, \mathcal{I}_{\mathcal{S}}$ in Eq. (2.19), respectively. Similarly, we denote the discrete version of the factor $\mathcal{F}_{\mathcal{Q}}, \mathcal{F}_{\mathcal{S}}$ in Eq. (2.19) as $\hat{\mathcal{F}}_{\mathcal{Q}}, \hat{\mathcal{F}}_{\mathcal{S}} : \mathbb{R} \to \mathbb{R}$, respectively.

a) For conservative systems (*i.e.*, $\mathcal{S}[R\,v] = 0$), the structure loss is designed as

$$\mathcal{L}_R[\bar{R}_{\mathrm{NN}}, \bar{v}_{\mathrm{copy}}] = \frac{1}{N_T}\sum_{n=0}^{N_T} \left| \hat{\mathcal{F}}_{\mathcal{Q}}[\bar{R}^n_{\mathrm{NN}}]\,\hat{\mathcal{I}}_{\mathcal{Q}}[\bar{v}^n_{\mathrm{copy}}] - C_0 \right|^2, \tag{4.4}$$

where the constant $C_0 = \mathcal{Q} \circ \iota[u_0]$ is given by the initial condition.

b) For dissipative systems (*i.e.*, $\mathcal{S}[R\,v] < 0$), we first apply the backward Euler method to discretize the structure ODE (2.19) as: for $n = 1, \cdots, N_T$,

$$\frac{\hat{\mathcal{F}}_{\mathcal{Q}}[R^n]\,\hat{\mathcal{I}}_{\mathcal{Q}}[v^n] - \hat{\mathcal{F}}_{\mathcal{Q}}[R^{n-1}]\,\hat{\mathcal{I}}_{\mathcal{Q}}[v^{n-1}]}{\delta t} = \hat{\mathcal{F}}_{\mathcal{S}}[R^n]\,\hat{\mathcal{I}}_{\mathcal{S}}[v^n]. \tag{4.5}$$

Combined with the initial condition $\mathcal{F}_{\mathcal{Q}}[R](0)\,\mathcal{I}_{\mathcal{Q}}[v](0) = C_0$, we denote the residual of the discrete structure ODE (4.5) as: for $n = 1, \cdots, N_T$,

$$\begin{cases} \mathcal{L}^0_R[R, v] := \left| \hat{\mathcal{F}}_{\mathcal{Q}}[R^0]\,\hat{\mathcal{I}}_{\mathcal{Q}}[v^0] - C_0 \right|^2, \\ \mathcal{L}^n_R[R, v] := \left| \dfrac{\hat{\mathcal{F}}_{\mathcal{Q}}[R^n]\,\hat{\mathcal{I}}_{\mathcal{Q}}[v^n] - \hat{\mathcal{F}}_{\mathcal{Q}}[R^{n-1}]\,\hat{\mathcal{I}}_{\mathcal{Q}}[v^{n-1}]}{\delta t} - \hat{\mathcal{F}}_{\mathcal{S}}[R^n]\,\hat{\mathcal{I}}_{\mathcal{S}}[v^n] \right|^2, \end{cases}$$

then the structure loss $\mathcal{L}_R$ can be designed using the residual $\mathcal{L}^n_R$ as:

$$\mathcal{L}_R[\bar{R}_{\mathrm{NN}}, \bar{v}_{\mathrm{copy}}] = \frac{1}{N_T}\sum_{n=0}^{N_T} \mathcal{L}^n_R[\bar{R}_{\mathrm{NN}}, \bar{v}_{\mathrm{copy}}]. \tag{4.6}$$

Moreover, inspired by the causal training strategy [86], we can reformulate the structure loss $\mathcal{L}_R$ as the weighted form to respect the temporal causality:

$$\begin{aligned} \widetilde{\mathcal{L}}_R[\bar{R}_{\mathrm{NN}}, \bar{v}_{\mathrm{copy}}] &= \frac{1}{N_T} \sum_{n=0}^{N_T} w_n \, \mathcal{L}_R^n[\bar{R}_{\mathrm{NN}}, \bar{v}_{\mathrm{copy}}], \\ \text{where} \quad w_n &= \exp\left(-\varepsilon \sum_{l=0}^{n-1} \mathcal{L}_R^l[\bar{R}_{\mathrm{NN}}, \bar{v}_{\mathrm{copy}}] \right). \end{aligned} \tag{4.7}$$

Remark 4.2. To discretize the structure ODE (2.19), we adopt the backward Euler method for simplicity, while it can be easily extended to other time discretization schemes, such as Runge-Kutta methods and backward difference formula methods.

Example 4.1 (Example 2.1 revisited)**.** For the Burgers' equation (2.9), we discrete the structure ODE (2.21) as: for $n = 1, \cdots, N_T$,

$$\frac{\left(R^{n+1}\right)^2 \hat{\mathcal{I}}_{\mathcal{Q}}[v^{n+1}] - (R^n)^2 \, \hat{\mathcal{I}}_{\mathcal{Q}}[v^n]}{\delta t} + 2\nu (R^{n+1})^2 \, \hat{\mathcal{I}}_{\mathcal{S}}[v^{n+1}] = 0,$$

then we denote the residual of the discrete structure ODE as: for $n = 1, \cdots, N_T$,

$$\begin{cases} \mathcal{L}_R^0[R, v] := \left| (R^0)^2 \, \hat{\mathcal{I}}_{\mathcal{Q}}[v^0] - \displaystyle\int_{-1}^{1} u_0^2(x) \, \mathrm{d}x \right|^2, \\ \mathcal{L}_R^n[R, v] = \left| \dfrac{(R^n)^2 \, \hat{\mathcal{I}}_{\mathcal{Q}}[v^n] - (R^{n-1})^2 \, \hat{\mathcal{I}}_{\mathcal{Q}}[v^{n-1}]}{\delta t} + 2\nu (R^n)^2 \, \hat{\mathcal{I}}_{\mathcal{S}}[v^n] \right|^2. \end{cases}$$

The structure loss $\widetilde{\mathcal{L}}_R[\bar{R}_{\mathrm{NN}}, \bar{v}_{\mathrm{copy}}]$ is then formulated by using the residual $\mathcal{L}_R^n$ as defined in Eq. (4.7).

4.2.3 Training Procedure

Although the primary-copilot design of Sidecar facilitates flexible integration with existing NN solvers, it also introduces significant training challenges: Two separate networks must serve different purposes while maintaining consistency, all without incurring excessive computational cost. We design the training procedure

of Sidecar to ensure that the structure-preserving knowledge can enhance, rather than constrain, the learning process. It involves two stages:

a) *Synchronization*: We equip the primary network $\bar{v}_{\text{NN}}(\mathbf{x}, t)$ with the copilot network $\bar{R}_{\text{NN}}(t)$, and train both networks to minimize the solver loss $\mathcal{L}_{\text{solver}}[\bar{R}_{\text{NN}}\, \bar{v}_{\text{NN}}]$ as defined in Eq. (4.3). During this stage, all training techniques inherited from the chosen NN solver can be applied, such as the adaptive sampling strategy [28,52] and the causal training strategy [86].

This stage allows the primary network $\bar{v}_{\text{NN}}(\mathbf{x}, t)$ to achieve sufficient accuracy, allowing a well-estimated spatial integration within the structure loss $\mathcal{L}_R$ in Eq. (4.4) or Eq. (4.6). Additionally, this stage ensures that the copilot network $\bar{R}_{\text{NN}}(t)$ is synchronized with the primary network $\bar{v}_{\text{NN}}(\mathbf{x}, t)$, offering a well-prepared initialization for the next stage.

b) *Navigation*: The solver loss $\mathcal{L}_{\text{solver}}[\bar{R}_{\text{NN}}\, \bar{v}_{\text{NN}}]$ continues to be minimized with respect to both $\bar{R}_{\text{NN}}(t)$ and $\bar{v}_{\text{NN}}(\mathbf{x}, t)$. Additionally, the structure loss $\mathcal{L}_R[\bar{R}_{\text{NN}}]$ is introduced and minimized with respect to the copilot network $\bar{R}_{\text{NN}}(t)$ only.

The second stage aims to navigate the learned solution $\bar{R}_{\text{NN}}(t)\, \bar{v}_{\text{NN}}(\mathbf{x}, t)$ to better satisfy the structure ODE (2.19), enhancing both accuracy and physical consistency. As discussed in Section 4.2.2, the spatial integration $\mathcal{I}_{\mathcal{Q}}[\bar{v}_{\text{copy}}]$ and $\mathcal{I}_{\mathcal{S}}[\bar{v}_{\text{copy}}]$ within the structure loss $\mathcal{L}_R$ are computed using a detached copy $\bar{v}_{\text{copy}}$, which is not involved in the back-propagation process. This stage acts as a fine-tuning process and thus require significantly fewer epochs compared to the first stage. If the first stage involves K_1 training epochs, the second stage typically uses $K_2 \ll K_1$.

The two-stage training is easy to implement by setting the coefficient $\alpha = 0$ in Eq. (4.2) during the first stage and then updating it to $\alpha = 1$ for the second stage. The overall training procedure for Sidecar is outlined in Algorithm 3.

Algorithm 3: Training Procedure of the Sidecar Framework

Input: PDE system (2.4) and its structure equation (2.8), the primary NN solver and its loss function $\mathcal{L}_{\text{solver}}$, the structure loss $\mathcal{L}_R$ as Eq. (4.4) or Eq. (4.6), the training epochs K_1 and K_2.

Output: A trained primary NN solver $\bar{v}(\mathbf{x}, t)$ and a copilot network $\bar{R}(t)$.

`// Stage 1: Synchronization`

for $k = 1$ *to* K_1 **do**

  Train $\bar{v}_{\text{NN}}(\mathbf{x}, t)$ and $\bar{R}_{\text{NN}}(t)$ to minimize the solver loss $\mathcal{L}_{\text{solver}}[\bar{R}_{\text{NN}}\, \bar{v}_{\text{NN}}]$ as in Eq. (4.3).

`// Stage 2: Navigation`

for $k = 1$ *to* K_2 **do**

  Compute the structure loss $\mathcal{L}_R$ by detached copy $\bar{v}_{\text{copy}}(\mathbf{x}, t)$ of $\bar{v}_{\text{NN}}(\mathbf{x}, t)$.

  Train $\bar{v}_{\text{NN}}(\mathbf{x}, t)$ and $\bar{R}_{\text{NN}}(t)$ to minimize the total loss $\mathcal{L}_{\text{solver}}[\bar{R}_{\text{NN}}\, \bar{v}_{\text{NN}}] + \mathcal{L}_R[\bar{R}_{\text{NN}}, \bar{v}_{\text{copy}}]$ as in Eq. (4.2).

4.3 Experiments

This section presents experiments demonstrating the Sidecar framework's effectiveness in enhancing NN solvers with structure knowledge.

4.3.1 Experimental Setup

For each primary network, we have implemented the vanilla version and the Sidecar-enhanced version, and compared their performance in terms of the primary NN solver loss Eq. (4.3), the L^2 distance to the exact solution as the exact L^2-error as $\|\bar{u} - u\|_2$, and the structure-preserving L^∞-error as

$$\max_{t\in[0,T]} \big|\mathcal{Q}[\bar{u}](t) - \mathcal{Q}[u](t)\big|.$$

For a fair comparison, the total number of neurons and layers of the sidecar-enhanced version is kept the same as the vanilla version. The width of the primary

solver $\bar{v}_{\mathrm{NN}}(\mathbf{x}, t)$ and the copilot network $\bar{R}_{\mathrm{NN}}(t)$ are denoted as W_v and W_R, respectively, while the total width in the vanilla version as $W_v + W_R$. As the copilot network only depends on the temporal variable, the total number of parameters of the Sidecar-enhanced version is actually slightly smaller than the vanilla version.

Both vanilla PINNs and Sidecar-enhanced PINNs are trained with the same training data and hyperparameters. The training data $\{(x_j, 0)\}_{j=1}^{N_{\mathrm{IC}}}$ and $\{(x_k, t_k)\}_{k=1}^{N_{\mathrm{BC}}}$ are equally spaced collocation points for the initial and boundary conditions, respectively, while the PDE residual points $\{(x_i, t_i)\}_{i=1}^{N_{\mathrm{PDE}}} \in \Omega \times [0, T]$ are the corresponding collocation points in the inner domain Ω, *i.e.*, $N_{\mathrm{IC}} \cdot N_{\mathrm{BC}} = N_{\mathrm{PDE}}$. The test set used to evaluate the performance of the trained models is $2\times$refined from the training set. For the two-stage training procedure of the Sidecar framework, the training epochs of the compared vanilla version K_0 are the same as the sum of the two stages of the Sidecar-enhanced version, *i.e.*, $K_0 = K_1 + K_2$.

The code is implemented in Python with the `PyTorch` library [70], while it can be easily extended to other deep learning frameworks such as `JAX` [3]. The experiments are conducted on an NVIDIA A100 GPU. Each experiment is repeated 10 times with different random seeds, and the results are averaged over these runs. The shaded areas in the error plots represent the trust intervals with a confidence level of 95%. The detailed hyperparameters of the Sidecar framework are summarized in Table 4.1. The code and data are publicly available on GitHub at https://github.com/DanclaChen/Sidecar.

4.3.2 Dissipative System: Burgers' Equation

We first apply the Sidecar framework to the illustrative example of the Burgers' equation (2.9) with the viscosity coefficient $\nu = 0.1$, and compare the performance of the Sidecar-enhanced PINNs with the vanilla PINNs. The initial condition and the corresponding exact solution [89] are given as

$$u(x, 0) = \frac{2\pi\nu \sin(\pi x)}{2 + \cos(\pi x)} \quad \Longrightarrow \quad u(x, t) = \frac{2\pi\nu \sin(\pi x) e^{-\pi^2 \nu t}}{2 + \cos(\pi x) e^{-\pi^2 \nu t}}. \tag{4.8}$$

Table 4.1: The hyperparameter of the Sidecar framework

	Burgers' equation	**NLS equation**	**Allen-Cahn equation**
L_v	2	4	4
W_v	16 32 64 128 256	25 50 100 200	64 128 256
L_R	1	2	2
W_R	8	10	16
K_1	20,000	100,000	180,000
K_2	10,000	20,000	20,000
N_{IC}	128	512	512
N_{BC}	100	128	200
N_{PDE}	12,800	65,536	10240
α	10	1	1

The solution function is plotted in the top panel of Figure 4.2.

We implemented the vanilla PINNs [72] for the original Burgers' equation (2.9), and the Sidecar-enhanced PINNs for the transformed system Eq. (2.20) and Eq. (2.21) after applying $u(x,t) = R(t)\, v(x,t)$. The compared vanilla PINN is an MLP trained using the PINNs loss function Eq. (2.5), following the vanilla PINNs design [72]. As for the Sidecar-enhanced PINNs, the primary network $\bar{v}_{\text{NN}}(x,t)$ and the copilot network $\bar{R}_{\text{NN}}(t)$ are both parameterized by MLPs, while $\bar{R}_{\text{NN}}(t)$ has much fewer parameters than $\bar{v}_{\text{NN}}(x,t)$. The solver loss $\mathcal{L}_{\text{solver}}$ is designed as the PINNs loss function Eq. (2.5), while the structure loss $\mathcal{L}_R$ is derived based on the loss design of the dissipative system Eq. (4.7). The learned solutions, compared to the exact solution,

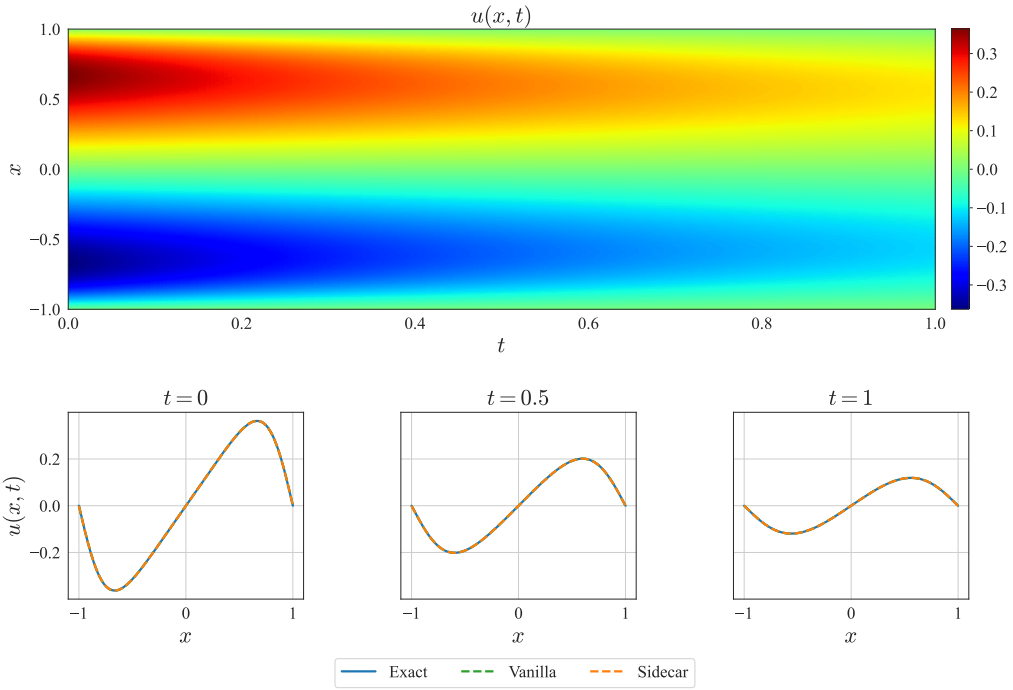

Figure 4.2: The smooth solution of the Burgers' equation. *Top*: The exact solution of the Burgers' equation. *Bottom*: Comparison of the exact solutions, the vanilla and Sidecar-enhanced PINNs solutions corresponding to the three temporal snapshots. The shown results are the worst cases of the 10 runs.

are shown in the bottom panel of Figure 4.2, while the error reduction with increasing network width is illustrated in Figure 4.3. The Sidecar-enhanced PINNs provide more accurate solutions and better preserve energy dissipation, demonstrating the framework's effectiveness in enhancing NN solvers with structure knowledge.

4.3.3 Conservation System: NLS Equation

We also apply the Sidecar framework to the 1D NLS equation in Eq. (2.11). Here we consider the moving soliton solution, a typical solution to the NLS equation describing a stable and localized wave packet that propagates without changing shape [19]. The initial condition and the corresponding exact solution are given as

$$u(x,0) = \operatorname{sech}(x)e^{-2ix} \quad \Longrightarrow \quad u(x,t) = \operatorname{sech}(x+2t)\, e^{-i(2x+\frac{3}{2}t)}. \tag{4.9}$$

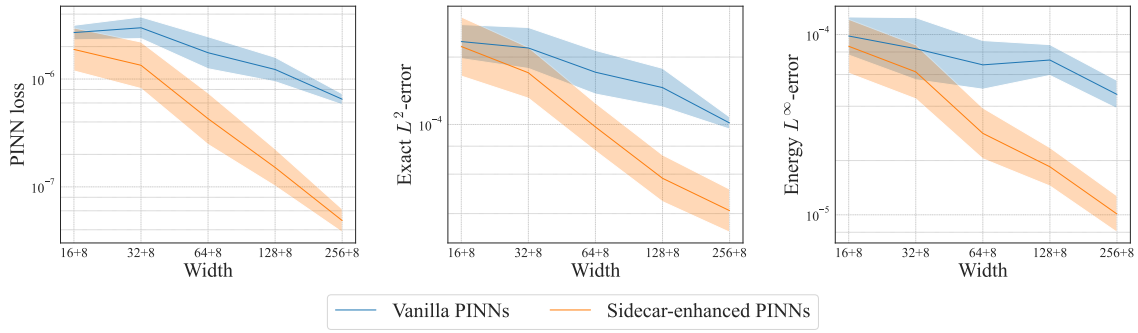

Figure 4.3: The comparison of the vanilla PINNs and the Sidecar-enhanced PINNs for the Burgers' equation.

Here, the spatial-temporal domain $(x,t) \in [-15,15] \times [0,\pi/2]$ is chosen to ensure that the soliton wave is fully captured. The solution function is plotted in the top panel of Figure 4.4.

Since the NLS equation is a complex-valued PDE system, *i.e.*, $u(x,t) \in \mathbb{C}$, the primary NN solver $v(x,t)$ is naturally a complex-value function, while the TDSR factor $R(t)$ could be either $R(t) \in \mathbb{C}$ or $R(t) \in \mathbb{R}$. Here, we choose a real-valued TDSR factor $R(t)$, and the overall solution can be written as

$$u(x,t) = R(t)\left(\mathrm{Re}[v](x,t) + i\ \mathrm{Im}[v](x,t)\right), \tag{4.10}$$

where $\mathrm{Re}[v]$ and $\mathrm{Im}[v]$ denote the real and imaginary parts of the complex function $v(x,t)$, respectively. It enables rewriting the norm of u as $|u|^2 = R^2\left(\mathrm{Re}[v]^2 + \mathrm{Im}[v]^2\right)$, and the real-valued form of the PDE system in Eq. (2.11) as

$$\begin{cases} -2R\ \mathrm{Im}[v]_t - 2R_t\ \mathrm{Im}[v] + R\ \mathrm{Re}[v]_{xx} + 2R^3\ \left(\mathrm{Re}[v]^2 + \mathrm{Im}[v]^2\right)\mathrm{Re}[v] = 0, \\ \ \ 2R\ \mathrm{Re}[v]_t + 2R_t\ \mathrm{Re}[v] + R\ \mathrm{Im}[v]_{xx} + 2R^3\ \left(\mathrm{Re}[v]^2 + \mathrm{Im}[v]^2\right)\mathrm{Im}[v] = 0, \end{cases} \tag{4.11}$$

where $(x,t) \in [-15,15] \times [0,\pi/2]$.

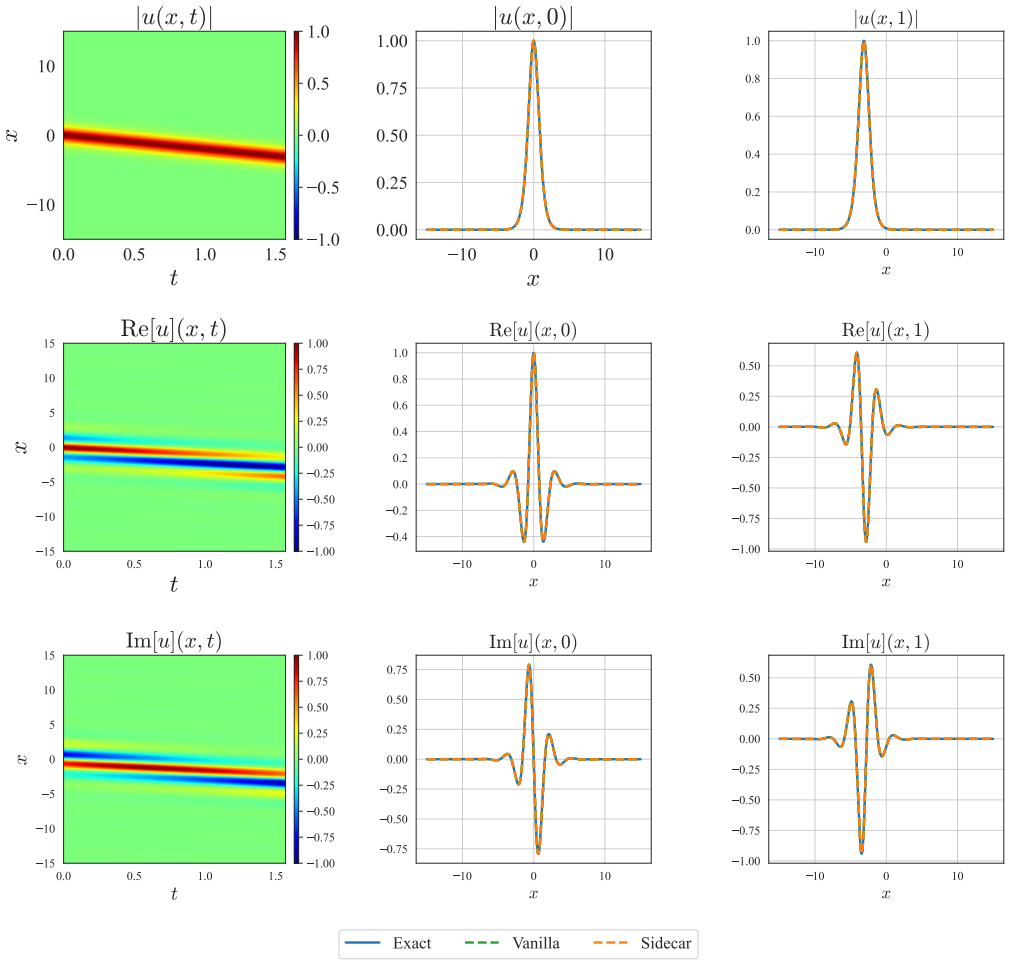

Figure 4.4: The smooth solution of the NLS equation. *Column 1*: The exact solution of the NLS equation. *Columns 2-3*: Comparison of the exact solutions, the vanilla and Sidecar-enhanced PINNs solutions corresponding to the three temporal snapshots. The shown results are the worst cases of the 10 runs.

Mass Conservation of NLS

We first consider the mass conservation law in Eq. (2.12), which is restated as

$$\mathcal{Q}_1[u](t) := \int_{-15}^{15} |u(x,t)|^2 \, \mathrm{d}x \equiv C_0^{(1)}, \quad \text{where} \quad C_0^{(1)} = \int_{-15}^{15} |u_0(x)|^2 \, \mathrm{d}x.$$

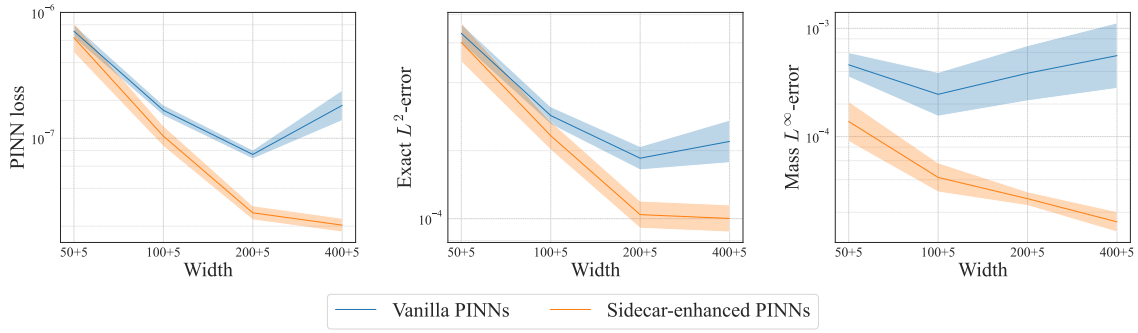

Figure 4.5: The comparison of the vanilla PINNs and the Sidecar-enhanced PINNs for the NLS equation with mass conservation.

After applying the transformation Eq. (4.10) and temporal discretization, the structure ODE of the mass conservation law gives

$$R^2\,\mathcal{I}_1[v] = C_1, \quad \text{where} \quad \mathcal{I}_1[v](t) = \int_{-15}^{15} |v(x,t)|^2\,\mathrm{d}x. \tag{4.12}$$

It is then used as the structure loss $\mathcal{L}_R$ for the conservative system (4.11). $(x,t) \in [-15,15] \times [0,\pi/2]$ leads to the mass constant $C_0^{(1)} = 2\tanh(15)$.

We implement the vanilla PINNs for the NLS equation (2.11), and the Sidecar-enhanced PINNs for the system Eq. (4.11) and Eq. (4.12) after applying the transformation in Eq. (4.10). The compared vanilla PINN is an MLP trained using the PINNs loss function Eq. (2.5), following the vanilla PINNs design [72]. As for the Sidecar-enhanced PINNs, the primary NN solver $\bar{v}_{\mathrm{NN}}(x,t)$ and the copilot network $\bar{R}_{\mathrm{NN}}(t)$ are parameterized by an MLP and a lightweight MLP, respectively. The solver loss $\mathcal{L}_{\mathrm{solver}}$ is designed as the PINNs loss function Eq. (2.5), while the structure loss $\mathcal{L}_R$ is derived from Eq. (4.12) following the conservative system Eq. (4.4), along with the causal training strategy Eq. (4.7). The results are shown in Figure 4.5.

Similar to the Burgers' equation, the Sidecar-enhanced PINNs result in more accurate solutions compared to the vanilla PINNs, while also better preserving the total mass of the system. Notably, as shown in the right panel of Figure 4.5, the numerical mass of the vanilla PINNs fails to converge as the network width increases.

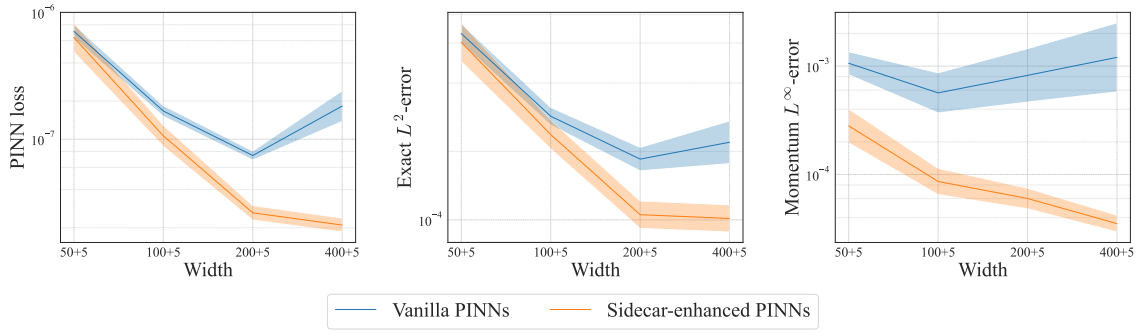

Figure 4.6: The comparison of the vanilla PINNs and the Sidecar-enhanced PINNs for the NLS equation with momentum conservation.

In contrast, the Sidecar-enhanced PINNs significantly improve the mass conservation property of the NLS equation.

Momentum Conservation of NLS

For the NLS equation, we can also consider the momentum conservation law in Eq. (2.13), which is restated as

$$\mathcal{Q}_2[u](t) \equiv C_0^{(2)}, \quad \text{where} \begin{cases} \mathcal{Q}_2[u](t) := \displaystyle\int_{-15}^{15} \mathrm{Re}[u]\ \mathrm{Im}[u_x] - \mathrm{Im}[u]\ \mathrm{Re}[u_x]\,\mathrm{d}x, \\ C_0^{(2)} = \displaystyle\int_{-15}^{15} \mathrm{Re}[u_0]\ \mathrm{Im}[(u_0)_x] - \mathrm{Im}[u_0]\ \mathrm{Re}[(u_0)_x]\,\mathrm{d}x. \end{cases}$$

After the transformation Eq. (4.10), the structure ODE of the momentum conservation law can be written as

$$R^2\,\mathcal{I}_2[v] = C_0^{(2)}, \text{ where } \mathcal{I}_2[v](t) = \int_{-15}^{15} \big(\mathrm{Re}[v]\ \mathrm{Im}[v_x] - \mathrm{Im}[v]\ \mathrm{Re}[v_x]\big)\,\mathrm{d}x. \tag{4.13}$$

Since $(x,t) \in [-15,15] \times [0,\pi/2]$, the momentum constant $C_0^{(2)} = -4\tanh(15)$.

Following the same setting as the mass conservation law, we compare the performance of vanilla PINNs with the Sidecar-enhanced PINNs. The only difference is that the structure loss $\mathcal{L}_R$ is derived from the structure ODE of momentum conservation law (4.13). The results shown in Figure 4.6 indicate that the Sidecar

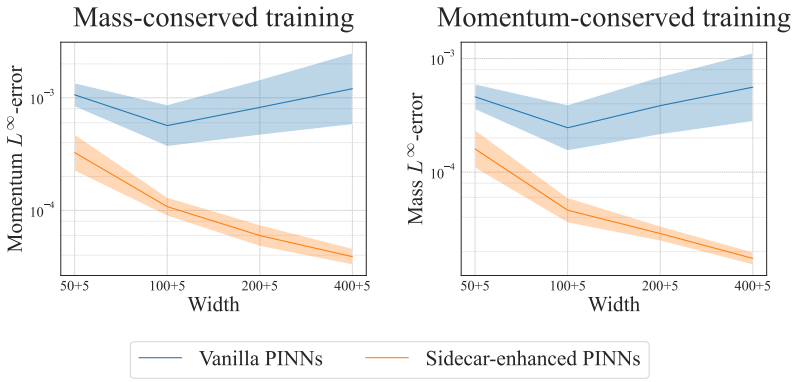

Figure 4.7: The comparison of the momentum/mass conservation performance for the NLS equation between the vanilla PINNs and the Sidecar-enhanced PINNs trained with the mass/conservation law only.

enhances both the solution accuracy and momentum conservation performance by incorporating the momentum conservation law.

Moreover, since the NLS equation has both mass and momentum conservation laws, we can further investigate the momentum conservation performance of the Sidecar-enhanced PINNs trained with the mass conservation law only (and vice versa). The results are shown in Figure 4.7. We can see that the Sidecar-enhanced PINNs, trained with the mass conservation law only, can also effectively improve momentum conservation. Future work includes exploring the preservation of multiple physical properties simultaneously.

4.3.4 The Allen-Cahn Equation

We also apply the Sidecar framework to the Allen-Cahn equation in Eq. (2.14). Here we follow the scenario in [86] to consider $\varepsilon = 0.01$ and $f[u] = 5(u - u^3)$, along with the initial condition as $u_0(x) = x^2 \cos(\pi x)$. The exact solution is not available for the Allen-Cahn equation, thus, we compute a high-resolution reference solution with a spectral method [78] as the ground truth. The solution function is plotted in the top panel of Figure 4.8.

Apply the transformation $u(x,t) = R(t)\, v(x,t)$, the original Allen-Cahn equation

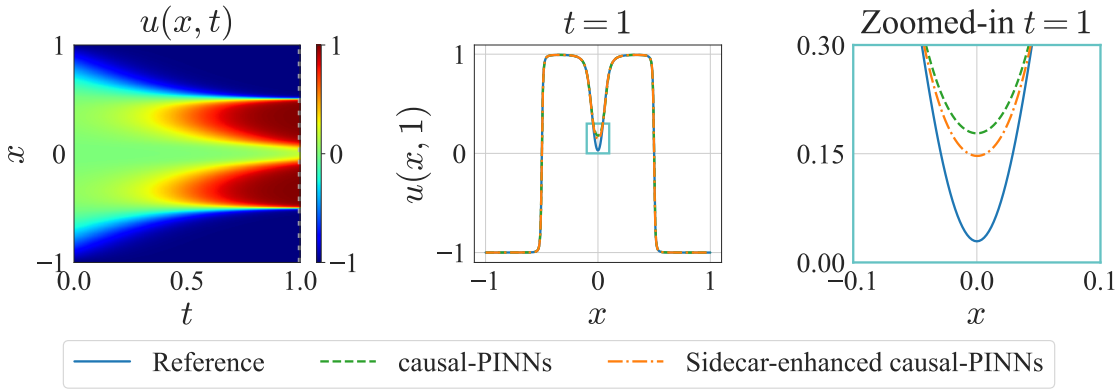

Figure 4.8: The solutions of the 1D AC equation (2.14). Left: Illustration of the reference solution. Middle: Comparison of the snapshot at $t = 1$ from the reference solution, the Sidecar-enhanced causal-PINNs, and the equivalent causal-PINNs solutions [86]. The cyan box indicates the region where the solution is zoomed in on the right. Right: The zoomed-in view of the solutions snapshot. The shown results are the worst cases of the 10 runs.

(2.14) and its energy dissipation law Eq. (2.15) can be rewritten as

$$\begin{cases} R\,v_t + R_t\,v = \varepsilon^2 R\,v_{xx} + f[R\,v], \\ \dfrac{\mathrm{d}}{\mathrm{d}t}\mathcal{E}_{AC}[R\,v] = \mathcal{S}_{AC}[R\,v]. \end{cases} \tag{4.14}$$

We simplify the energy functional $\mathcal{E}_{AC}[R\,v]$ and the dissipation speed $\mathcal{S}_{AC}[R\,v]$ by factoring out R from integration as

$$\begin{aligned} \mathcal{E}_{AC}[R\,v] &= R^4 \int_{-1}^{1} \frac{5}{4} v^4 \,\mathrm{d}x + R^2 \int_{-1}^{1} \left(\frac{\varepsilon^2}{2} v_x^2 - \frac{5}{2} v^2 \right) \mathrm{d}x + \int_{-1}^{1} \frac{5}{4} \,\mathrm{d}x, \\ \mathcal{S}_{AC}[R\,v] &= -R_t^2 \int_{-1}^{1} v^2 \,\mathrm{d}x - R^2 \int_{-1}^{1} v_t^2 \,\mathrm{d}x - 2R_t R \int_{-1}^{1} v v_t \,\mathrm{d}x. \end{aligned}$$

By omitting the constant term, the structure ODE of the energy dissipation law can be derived as

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\left(R^4\,\mathcal{I}_{\mathcal{Q},1}[v] + R^2\,\mathcal{I}_{\mathcal{Q},2}[v] \right) = R_t^2\,\mathcal{I}_{\mathcal{S},1}[v] + R^2\,\mathcal{I}_{\mathcal{S},2}[v] + RR_t\,\mathcal{I}_{\mathcal{S},3}[v], \\ R^4(0)\,\mathcal{I}_{\mathcal{Q},1}[v](0) + R^2(0)\,\mathcal{I}_{\mathcal{Q},2}[v](0) = C_0, \end{cases} \tag{4.15}$$

where

$$\mathcal{I}_{\mathcal{Q},1}[v] = \frac{5}{4}\int_{-1}^{1} v^4 \,\mathrm{d}x, \quad \mathcal{I}_{\mathcal{Q},2}[v] = \int_{-1}^{1}\left(\frac{\varepsilon^2}{2}v_x^2 - \frac{5}{2}v^2\right)\mathrm{d}x, \quad C_0 = \mathcal{E}_{AC}\circ\iota[u_0],$$
$$\mathcal{I}_{\mathcal{S},1}[v] = -\int_{-1}^{1} v_t^2 \,\mathrm{d}x, \quad \mathcal{I}_{\mathcal{S},2}[v] = -\int_{-1}^{1} v^2 \,\mathrm{d}x, \quad \mathcal{I}_{\mathcal{S},3}[v] = -2\int_{-1}^{1} v v_t \,\mathrm{d}x.$$

After temporal discretization with the backward Euler scheme Eq. (4.5), we denote the residual of the structure ODE as: for $n = 1, 2, \ldots, N_T$,

$$\begin{cases} \mathcal{L}_R^0 = \left| \left(R^0\right)^4 \mathcal{I}_{\mathcal{Q},1}[v^0] + \left(R^0\right)^2 \mathcal{I}_{\mathcal{Q},2}[v^0] - C_0 \right|^2, \\ \mathcal{L}_R^n = \Big| \left(\left(R^n\right)^4 \mathcal{I}_{\mathcal{Q},1}[v^n] + \left(R^n\right)^2 \mathcal{I}_{\mathcal{Q},2}[v^n] \right) \\ \qquad - \left(\left(R^{n-1}\right)^4 \mathcal{I}_{\mathcal{Q},1}[v^{n-1}] + \left(R^{n-1}\right)^2 \mathcal{I}_{\mathcal{Q},2}[v^{n-1}] \right) \\ \qquad - \delta t \left(\left(\widetilde{R}_t^n\right)^2 \mathcal{I}_{\mathcal{S},1}[v^n] + \left(R^n\right)^2 \mathcal{I}_{\mathcal{S},2}[v^n] + \widetilde{R}_t^n R^n \mathcal{I}_{\mathcal{S},3}[v^n] \right) \Big|^2, \end{cases} \tag{4.16}$$

where $\widetilde{R}_t^n = (R^n - R^{n-1})/\delta t$ is the difference quotient of $R(t)$.

Remark 4.3. The discrete structure ODE (4.15) for the Allen-Cahn equation (2.14) has a more complicated form, mainly due to the dissipation speed $\mathcal{S}_{AC}[R\,v]$ in Eq. (2.16) involving the temporal derivative of $R(t)$.

Experimental setting: We follow the causal training strategy [86] to reformulate the PDE loss $\mathcal{L}_{\mathrm{PDE}}$ in Eq. (2.5) as $\widetilde{\mathcal{L}}_{\mathrm{PDE}}$ in Eq. (2.7), resulting in a variant of vanilla PINNs [72] (referred to as CausalPINNs) for the Allen-Cahn equation. The CausalPINNs model is implemented as an MLP, with the loss function defined as the sum of the causal PDE loss and the data loss related to the initial and boundary conditions, *i.e.*, $\mathcal{L}_{\mathrm{solver}} = \widetilde{\mathcal{L}}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{data}}$.

We apply CausalPINNs to the original Allen-Cahn equation (2.14) and the Sidecar-enhanced CausalPINNs to the transformed system (4.14) and (4.15) after introducing $u(x,t) = R(t)\,v(x,t)$. The CausalPINNs model is implemented as an MLP, while the Sidecar-enhanced CausalPINNs consist of an MLP for the primary solver and a lightweight MLP for the copilot network. The solver loss $\mathcal{L}_{\mathrm{solver}}$ follows

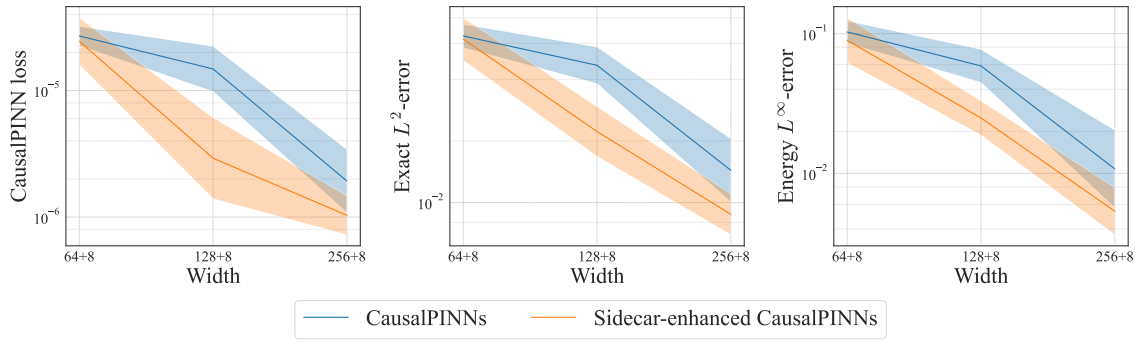

Figure 4.9: The comparison of the CausalPINNs and the Sidecar-enhanced CausalPINNs for the Allen-Cahn equation.

the CausalPINNs loss function $\mathcal{L}_{\text{solver}} = \widetilde{\mathcal{L}}_{\text{PDE}} + \mathcal{L}_{\text{data}}$. The structure loss $\mathcal{L}_R$ is derived from the structure ODE residual Eq. (4.16), following the dissipative system formulation in Eq. (4.6).

The results are shown in Figure 4.9. Compared to the CausalPINNs, the Sidecar-enhanced CausalPINNs achieve higher solution accuracy and better preservation of the energy dissipation property. This demonstrates that the Sidecar framework can also be integrated with other primary NN solvers, showcasing its flexibility and generality.

4.4 Further Discussions

In this section, we further discuss the reason why the Sidecar framework can enhance the performance of existing NN solvers for PDEs. A series of ablation studies is conducted to investigate the effectiveness of the main components in the Sidecar framework. Specifically, we are interested in whether the Sidecar framework benefits from:

1. improving the representation capacity of the NNs via the Sidecar architecture, or

2. incorporating the structure-preserving knowledge via the loss design.

We conduct a series of experiments to validate the above hypotheses, and the results show that the Sidecar framework can benefit from both ways.

4.4.1 The Representation Capacity of Sidecar Architecture

In the Sidecar framework, the copilot network $\bar{R}_{\mathrm{NN}}(t)$ only depends on t, as the structure-preserving properties are mainly related to the temporal evolution of preserved quantities. During training, the temporal-dependent features captured by $\bar{R}_{\mathrm{NN}}(t)$ could facilitate the learning of the primary NN solver $\bar{v}_{\mathrm{NN}}(x,t)$. Compared to the MLP in Eq. (2.1) used in vanilla PINNs, the Sidecar architecture may enhance the network's ability to represent PDE solutions with temporal evolution, thereby improving the performance of NN solvers. Here, we conduct an ablation study to validate this hypothesis.

Experimental setting: To evaluate the enhancement in representation capacity provided by the Sidecar architecture, we compare the approximation performance to the exact solution of networks adopted in vanilla and Sidecar-enhanced PINNs. Specifically, we consider an MLP equipped with a copilot network as $\bar{u}_{\mathrm{NN}}(x,t) = \bar{R}_{\mathrm{NN}}(t)\,\bar{v}_{\mathrm{NN}}(x,t)$, and a vanilla MLP $\bar{u}_{\mathrm{NN}}(x,t)$ with an equivalent total number of neurons and layers. Both NNs are trained to approximate the exact solution of the Burgers' equation (2.9) and the NLS equation (4.9) with the same training data and hyperparameters. The performance is compared in terms of the L^2 distance to the exact solution.

The results are shown in Figure 4.10, where the Sidecar-enhanced MLP consistently outperforms the vanilla MLP. This supports the hypothesis that the Sidecar architecture improves the representation capacity for PDE solutions with temporal evolution, thereby enhancing the performance of NN solvers.

With the same number of neurons and layers, the Sidecar-enhanced MLP has fewer parameters than the vanilla MLP while achieving more accurate approximations of the exact solution. Although increasing parameter numbers generally improves approximation accuracy, architectures specifically designed for particular

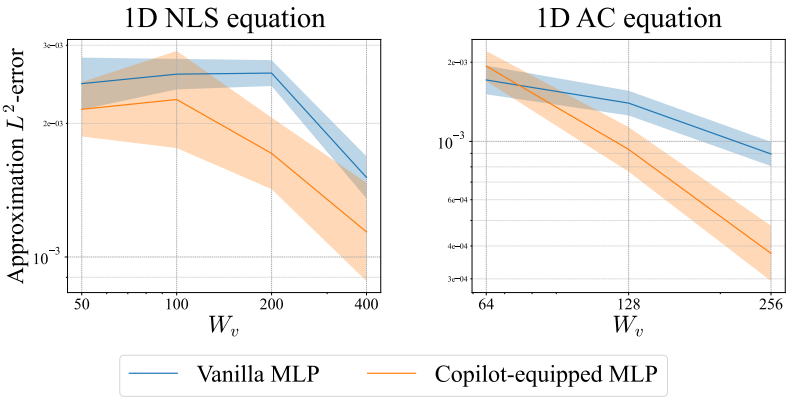

Figure 4.10: Comparison of copilot-equipped MLPs and the equivalent vanilla MLPs for approximating the reference solutions of the 1D NLS equation Eq. (2.11) and the 1D AC equation Eq. (2.14).

target functions can outperform standard designs. This principle is evident in the success of the Convolution Neural Networks (CNNs) for image processing [53], the Recurrent Neural Networks (RNNs) for sequential data [84], etc. Similarly, the Sidecar architecture can be viewed as a PDE-friendly design, tailored for PDE systems with temporal evolution.

In addition to the commonly used MLP architecture, other novel network designs have been proposed to enhance the performance of NN solvers. For example, a modified MLP has been derived based on gradient flow analysis [87], and a volume weighting method has been proposed to address the ill-conditioning of PDE losses [81]. These architectures involved additional connections to the vanilla MLP, sharing the same spirit as the Sidecar framework. However, these approaches do not explicitly incorporate structure-preserving knowledge, which is a key feature of the Sidecar framework.

4.4.2 The Effectiveness of the Loss Function Design and Implementation

After confirming the enhanced representation capacity provided by the Sidecar architecture, we now examine the effectiveness of the Sidecar loss design and implementation $\mathcal{L}_{\text{Sidecar}} = \mathcal{L}_{\text{solver}} + \alpha\mathcal{L}_R$ in Eq. (4.2). Although $\mathcal{L}_{\text{Sidecar}}$ follows the common "main loss + regularization term" format seen in existing structure-preserving NN solvers [42, 51], its design and implementation, particularly the structure loss $\mathcal{L}_R$ derived from the structure-preserving properties of the PDE system in Eq. (4.4) or Eq. (4.6), are uniquely tailored to the Sidecar framework.

To validate the effectiveness of $\mathcal{L}_{\text{Sidecar}}$, we consider a sufficient condition: the exact solution of the PDE system should minimize $\mathcal{L}_{\text{Sidecar}}$. Therefore, if the learned solution $\bar{R}\,\bar{v}$ is sufficiently accurate, it should remain stable when further trained with $\mathcal{L}_{\text{Sidecar}}[\bar{R}, \bar{v}]$. To verify this, we initialize the networks with the learned exact solution, and continue training with $\mathcal{L}_{\text{Sidecar}}$.

Experimental setting: We initialize the Sidecar-enhanced PINNs using the exact solution learned in Section 4.4.1. Then we follow the second *Navigation* stage of the Sidecar training procedure to train the networks with $\mathcal{L}_{\text{Sidecar}}$. We consider the exact solution of Burgers' equation (4.8), comparing the accuracy before and after training with $\mathcal{L}_{\text{Sidecar}}$. All hyperparameters remain consistent with those specified in Section 4.3.

The results are shown in Figure 4.11. For most random seeds, the L^2 distance to the exact solution remains stable while minimizing $\mathcal{L}_{\text{Sidecar}}$. Meanwhile, the PDE residual error and the structure-preserving properties of the learned exact solution are further improved by $\mathcal{L}_{\text{Sidecar}}$. These observations numerically demonstrate that the novel design and implementation of $\mathcal{L}_{\text{Sidecar}}$ align well with the PDE system.

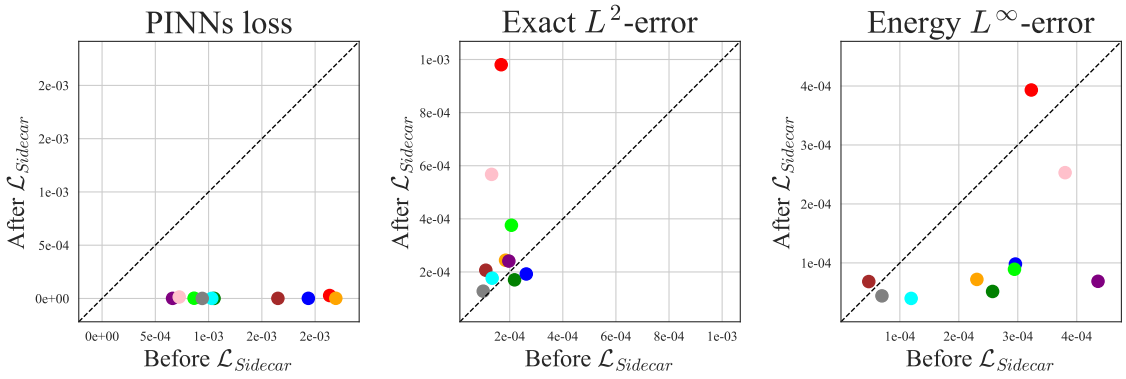

Figure 4.11: The comparison of the effectiveness of the Sidecar loss function design. Each point is the result of one random seed. The x- and y-axes represent the MSE error before and after training with the Sidecar loss $\mathcal{L}_{\text{Sidecar}}$, respectively. The point in the lower right corner corresponds to the case where the accuracy of the learned exact solution is further improved by the Sidecar loss $\mathcal{L}_{\text{Sidecar}}$.

4.4.3 The Necessity of the Structure Loss $\mathcal{L}_R$

Here, we validate the necessity of the structure loss $\mathcal{L}_R$ in incorporating structure-preserving knowledge. Ideally, $\mathcal{L}_R$ should complement the PDE-based solver loss $\mathcal{L}_{\text{solver}}$ by explicitly embedding structure-preserving properties into the training process. However, since these properties are inherently consistent with the PDE formulation, it is possible that improvements in structure-preserving performance could be achieved using $\mathcal{L}_{\text{solver}}$ alone. To investigate this, we compare the performance of the Sidecar-enhanced PINNs with and without the inclusion of $\mathcal{L}_R$.

Experimental setting: We evaluate the NLS equation (2.11), comparing the Sidecar-enhanced PINNs with and without the structure loss $\mathcal{L}_R$ during the second Navigation stage. All other settings follow Table 4.1.

The results are shown in Figure 4.12. Although the PDE residual error and the squared L^2 distance to the exact solution are comparable for the Sidecar-enhanced PINNs with or without the structure loss $\mathcal{L}_R$, the preservation of the system's considered quantities is improved when $\mathcal{L}_R$ is included, particularly for larger network widths. This highlights the critical role of $\mathcal{L}_R$ in embedding structure-preserving knowledge into the training process.

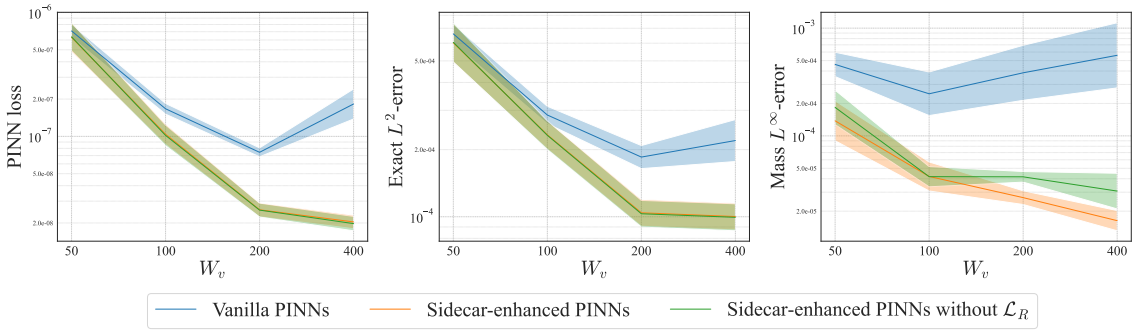

Figure 4.12: Comparison of the Sidecar-enhanced PINNs with and without the structure loss $\mathcal{L}_R$, and the equivalent vanilla PINNs [72] for 1D NLS equation Eq. (2.11). From left to right: the PINN test loss, the L^2 error of the numerical solution, and the L^∞ error of the mass conservation.

The added structure loss $\mathcal{L}_R$ in our Sidecar framework shares a similar spirit with the regularization terms used in the existing structure-preserving NN solvers [42,51], which aim to enforce intrinsic physical properties of the PDE system during training. However, these methods often suffer from performance degradation, as the added regularization terms can create an unreasonable trade-off between solution accuracy and physical fidelity. In contrast, the Sidecar framework integrates structure-preserving knowledge into the training process in a way that enhances physical consistency without sacrificing solution accuracy.

Chapter 5

Conclusions and Future Research

This dissertation aims to incorporate the structure-preserving concepts into the design and implementation of NNs, considering both the theoretical and applied aspects. The main contributions of this dissertation are summarized as follows:

1. We prove the UAP of permutation-trained NNs with random initialization for one-dimensional continuous functions.

2. We propose a novel structure-preserving framework, Sidecar, for enhancing the accuracy and physical fidelity of existing NN solvers.

The first part of this dissertation focuses on a novel method, permutation training, which exhibits unique properties and practical potential. To verify its efficacy, we prove the UAP of permutation-trained networks with random initialization for one-dimensional continuous functions. The proof is generalized from the equidistant scenario, where the key idea involves a four-pair construction of step function approximators, along with a processing method to eliminate the impact of the remaining parameters. Our numerical experiments not only confirm the theoretical results but also validate the prevalence of the UAP of permutation-trained networks in various initializations. The discovery that commonly used initializations fail to achieve UAP also raises an intriguing question about the systematic characterization of initializations that satisfy UAP. Our observation suggests that permutation

training could be a novel tool to describe the network learning behavior. Overall, we believe that the UAP of permutation-trained networks reveals the profound, yet undiscovered, insights into how the weight encodes the learned information, highlighting the importance of further theoretical and practical exploration.

The second part of this dissertation introduces Sidecar, a structure-preserving framework designed to enhance existing NN solvers. The framework combines a primary NN solver with a lightweight copilot network, trained jointly to minimize a PDE-based solver loss $\mathcal{L}_{\text{solver}}$ and a structure loss $\mathcal{L}_R$. The structure loss explicitly incorporates the system's structure-preserving properties, ensuring solutions adhere to intrinsic physical laws. A two-stage training procedure is employed to first synchronize the networks and then navigate the learned solution to respect these properties. The Sidecar framework is flexible, compatible with existing NN solvers, and applicable to a wide range of PDE systems with different structure-preserving properties. Experiments on the Burgers' equation, the NLS equation, and the Allen-Cahn equation demonstrate the Sidecar framework's effectiveness in improving solution accuracy and physical fidelity. The Sidecar-enhanced PINNs outperform vanilla PINNs in solution accuracy while better preserving system properties like energy dissipation, mass conservation, and momentum conservation. Ablation studies further validate the framework's key components, showing improvements in representation capacity, the effectiveness of the loss function design, and the necessity of the structure loss $\mathcal{L}_R$ for embedding additional knowledge. We believe that the Sidecar framework offers a promising approach to improving NN solvers for PDEs by leveraging structure-preserving principles.

Although the two main parts of this dissertation address different facets of structure preservation in NNs, they are unified by a common goal: advancing both the theoretical understanding and practical capabilities of NNs through explicit consideration of underlying system structures. In the first part, the preserved structure is the permutation invariance of the training process, which leads to a novel training method with theoretical guarantees and practical potential. The second part

focuses on the preservation of physical structures in PDEs, resulting in a framework that improves both the accuracy and physical fidelity of NN solvers. By integrating classical concepts from approximation theory and numerical analysis with contemporary machine learning methodologies, this dissertation charts a promising path for future research in the field of NNs and their applications.

1. Future Directions of Permutation Training

Our UAP results for permutation training in Chapter 3 provide the first theoretical guarantee (to our knowledge) for the effectiveness of this novel training method. However, the setting is still basic since it only considers one-dimensional continuous functions and shallow MLPs without estimation of the approximation rate, which is mainly due to that the unique setting of permutation training hinders the proof construction. The randomness in the initialization also necessitates advanced techniques in stochastic analysis, which are not fully explored in this work. Here, we briefly discuss some intriguing questions about generalization raised for future research.

Generalizing to Other Scenarios

Although we mainly focus on basic settings, the proof idea exhibits generalizability. However, extending our theoretical results to the high-dimensional scenario still faces challenges. One of the primary obstacles is constructing multi-dimensional basis functions that are suitable for the permutation training scenario. A reasonable approach relies on the construction in the finite element methods [4]. We plan to address this problem in future work. As for the numerical evidence of high-dimensional scenarios, [71] has examined the classification problem using VGG-based networks on the CIFAR-10 dataset, while our experiments in regression tasks have shown approximation behavior for two and three-dimensional inputs (see Section 3.4.4).

Obtaining an error bound in terms of the network width and depth is another intriguing question. Existing proof techniques typically rely on the continuity of the

target function, such that the approximation error of a given network to the target function with a certain level of continuity can be bounded by the network width and depth [80, 95]. Similarly, the continuity of the target function also influences our proof construction in Section 3.3, especially in Step 1, where the target function is approximated by a piecewise constant function. By introducing the modulus of continuity (like in [80]), an error bound regarding the network width is expected. However, deriving tighter or optimal error bounds may require a more systematic and refined approach to the proof construction.

Another interesting direction is to establish the UAP for deeper networks. In a free training scenario, the UAP results of shallow networks can be readily extended to deeper networks by constructing identity functions in the following layers, or rewriting a deeper network as a shallow network with more width [80]. Our linear reorganization in Eq. (3.14) enables construct an identity function $y = x$ using a pair of basis functions $y = p_n\phi_1^+(x) + q_n\phi_1^-(x)$, where $b_1 = 0$, $p_n = 1$, $q_n = -1$. This process enables us to utilize identity functions within subsequent layers. However, estimating the influence of the network depth on the UAP of permutation training is still an open question.

Permutation Training as a Theoretical Tool

Our observation in Section 3.4.8 indicates the theoretical potential of permutation training, as it corresponds well with the training process and has systematic mathematical descriptions. Specifically, the patterns observed in Figure 3.9 can intuitively lead to certain weight categorization strategies, potentially benefit consolidating the crucial weights for previous tasks [64], or pruning to find the ideal subnetwork in the lottery ticket hypothesis [26]. Additionally, the existing permutation training algorithm shares the same form as weight projection methods in continual learning [93], as it can be viewed as applying an order-preserving projection from the free training results to the initial weight value.

The Algorithmic Implementation

This work is expected to facilitate the applications of permutation training. However, some practical issues still exist and deserve further investigation. As a preliminary attempt, the existing permutation training algorithm, LaPerm, guides the permutation by inner loops, thus incurring undesirable external computation costs. However, employing more advanced and efficient search approaches, such as the learn-to-rank formalism [9], or permutation search algorithms in the study of LMC [1, 44], the benefits of permutation training will be actualized in practice. Importantly, our proof does not rely on any specific algorithmic implementations. Additionally, the incompatible initialization issue plotted in Figure 3.3(b) emphasizes the need for developing more effective initializations as well as investigating the criterion of UAP-compatible initializations.

2. Future Directions of Sidecar

As a versatile framework, Sidecar can be applied to various PDEs with different structure-preserving properties. Here is a brief discussion of potential future directions.

Extending to More Complex PDEs

Future work will extend the Sidecar framework to more complex and high-dimensional PDEs, such as high-dimensional Allen-Cahn equations or Navier-Stokes equations. Thanks to the additional structure information provided by the Sidecar framework, we can also explore the possibility of accelerating the training process in larger-scale problems.

Exploring Local Structure Properties

The current implementation of Sidecar focuses on global quantities of interest, such as energy dissipation, mass conservation, and momentum conservation.

However, many other structure-preserving properties exist in the form of local constraints, such as the divergence-free condition in incompressible Navier-Stokes equations, or the local energy dissipation law in the Cahn-Hilliard equation [67]. Random sampling strategies can be employed to incorporate these local constraints into the Sidecar framework.

Preserving Multiple Structure-Preserving Properties

It's also worth exploring the preservation of multiple structure-preserving properties simultaneously, which can be achieved by designing a more sophisticated structure loss function or using a multi-task learning approach. For example, in the case of NLS equations, we can consider the preservation of both mass and momentum conservation properties.

Integrating with Operator Learning

Integrating Sidecar with the evolution-operator learning methods like FNO [60] to enhance physical fidelity during temporal evolution is another promising direction. There are also some recent works that incorporate structure-preserving into operator learning methods, such as [94]. This approach is based on the scalar auxiliary variable (SAV) method [79], sharing the same idea of Lagrange multipliers as TSDR. Therefore, the Sidecar framework can be easily integrated with these operator learning methods to facilitate the training process.

References

[1] S. Ainsworth, J. Hayase, and S. Srinivasa. Git Re-Basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023.

[2] S. M. Allen and J. W. Cahn. A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. *Acta metallurgica*, 27(6):1085–1095, 1979.

[3] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[4] S. C. Brenner and L. R. Scott. *The mathematical theory of finite element methods*. Springer, 2008.

[5] Y. Cai. Achieve the minimum width of neural networks for universal approximation. In *The Eleventh International Conference on Learning Representations*, 2023.

[6] Y. Cai, G. Chen, and Z. Qiao. Neural networks trained by weight permutation are universal approximators. *Neural Networks*, page 107277, 2025.

[7] Z. Cai, J. Chen, and M. Liu. Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation. *Journal of Computational Physics*, 443:110514, 2021.

[8] P. J. Cameron. *Permutation groups*. Number 45. Cambridge University Press, 1999.

[9] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[10] J. L. Castro, C. J. Mantas, and J. Benıtez. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks*, 13(6):561–563, 2000.

[11] G. Chen, L. Ju, and Z. Qiao. A structure-preserving framework for solving parabolic partial differential equations with neural networks. *arXiv preprint arXiv:2504.10273*, 2025.

[12] S. Chen and G. D. Doolen. Lattice Boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1):329–364, 1998.

[13] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[14] B. Chenglong, L. Qianxiao, S. Zuowei, C. Tai, L. Wu, and X. Xueshuang. Approximation analysis of convolutional neural networks. *East Asian Journal on Applied Mathematics*, 13(3):524–549, 2023.

[15] S. H. Christiansen, H. Z. Munthe-Kaas, and B. Owren. Topics in structure-preserving discretization. *Acta Numerica*, 20:1–119, 2011.

[16] T. Cohen and M. Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[17] J. T. Cole and Z. H. Musslimani. Time-dependent spectral renormalization method. *Physica D: Nonlinear Phenomena*, 358:15–24, 2017.

[18] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[19] L. Debnath. *Nonlinear partial differential equations for scientists and engineers*. Springer, 2005.

[20] Q. Du, L. Ju, X. Li, and Z. Qiao. Maximum bound principles for a class of semilinear parabolic equations and exponential time-differencing schemes. *SIAM Review*, 63(2):317–359, 2021.

[21] Y. Duan, L. Li, G. Ji, and Y. Cai. Vanilla feedforward neural networks as a discretization of dynamical systems. *Journal of Scientific Computing*, 101(3):82, 2024.

[22] R. Entezari, H. Sedghi, O. Saukh, and B. Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations*, 2021.

[23] F. Fan, J. Xiong, and G. Wang. Universal approximation with quadratic deep networks. *Neural Networks*, 124:383–392, 2020.

[24] J. Feldmann, N. Youngblood, M. Karpov, H. Gehring, X. Li, M. Stappers, M. Le Gallo, X. Fu, A. Lukashchuk, A. S. Raja, et al. Parallel convolutional processing using an integrated photonic tensor core. *Nature*, 589(7840):52–58, 2021.

[25] W. Feller. *An introduction to probability theory and its applications, volume I*. John Wiley & Sons, 3 edition, 1968.

[26] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations*, 2019.

[27] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.

[28] Z. Gao, L. Yan, and T. Zhou. Failure-informed adaptive sampling for PINNs. *SIAM Journal on Scientific Computing*, 45(4):A1971–A1994, 2023.

[29] W. Gautschi. *Numerical analysis*. Springer Science & Business Media, 2011.

[30] Y. Geng, Y. Teng, Z. Wang, and L. Ju. A deep learning method for the dynamics of classic and conservative Allen-Cahn equations based on fully-discrete operators. *Journal of Computational Physics*, 496:112589, 2024.

[31] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[32] P. Gómez, H. H. Toftevaag, and G. Meoni. torchquad: Numerical integration in arbitrary dimensions with pytorch. *Journal of Open Source Software*, 6(64):3439, 2021.

[33] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[34] R. Guo, S. Cao, and L. Chen. Transformer meets boundary value inverse problems. In *The Eleventh International Conference on Learning Representations*, 2023.

[35] J. He, L. Li, and J. Xu. Approximation properties of deep ReLU CNNs. *Research in the mathematical sciences*, 9(3):38, 2022.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[37] Q. Hernández, A. Badías, D. González, F. Chinesta, and E. Cueto. Structure-preserving neural networks. *Journal of Computational Physics*, 426:109950, 2021.

[38] J. E. Hopcroft, J. D. Ullman, and A. V. Aho. *Data structures and algorithms*, volume 175. Addison-wesley Boston, MA, USA:, 1983.

[39] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[40] D. Hou, L. Ju, and Z. Qiao. Energy-dissipative spectral renormalization exponential integrator method for gradient flow problems. *SIAM Journal on Scientific Computing*, 46(6):A3477–A3502, 2024.

[41] J. Huang, C. Guestrin, and L. Guibas. Fourier theoretic probabilistic inference over permutations. *Journal of machine learning research*, 10(5), 2009.

[42] Q. Huang, J. Ma, and Z. Xu. Mass-preserving spatio-temporal adaptive PINN for Cahn-Hilliard equations with strong nonlinearity and singularity. *arXiv preprint arXiv:2404.18054*, 2024.

[43] T. W. Hungerford. *Algebra*, volume 73. Springer Science & Business Media, 2012.

[44] K. Jordan, H. Sedghi, O. Saukh, R. Entezari, and B. Neyshabur. REPAIR: Renormalizing permuted activations for interpolation repair. In *The Eleventh International Conference On Representation Learning*, 2023.

[45] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[46] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

[47] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[48] A. Kosuge, M. Hamada, and T. Kuroda. A 16 nj/classification fpga-based wired-logic DNN accelerator using fixed-weight non-linear neural net. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4):751–761, 2021.

[49] A. Kosuge, Y.-C. Hsu, M. Hamada, and T. Kuroda. A 0.61-μj/frame pipelined wired-logic DNN processor in 16-nm FPGA using convolutional non-linear neural network. *IEEE Open Journal of Circuits and Systems*, 3:4–14, 2021.

[50] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[51] M. Kütük and H. Yücel. Energy dissipation preserving physics informed neural network for Allen-Cahn equations. *arXiv preprint arXiv:2411.08760*, 2024.

[52] C. L. Wight and J. Zhao. Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. *Communications in Computational Physics*, 29(3):930–954, 2021.

[53] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[54] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[55] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019.

[56] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[57] R. J. LeVeque. *Numerical methods for conservation laws*, volume 132. Springer, 1992.

[58] Q. Li, T. Lin, and Z. Shen. Deep learning via dynamical systems: An approximation perspective. *Journal of the European Mathematical Society*, 25(5):1671–1709, 2022.

[59] Q. Li, T. Lin, and Z. Shen. Deep neural network approximation of invariant functions through dynamical systems. *Journal of Machine Learning Research*, 25(278):1–57, 2024.

[60] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[61] J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[62] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[63] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. volume 30, 2017.

[64] D. Maltoni and V. Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, 2019.

[65] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019.

[66] R. Molinaro, Y. Yang, B. Engquist, and S. Mishra. Neural inverse operators for solving PDE inverse problems. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 25105–25139. PMLR, 23–29 Jul 2023.

[67] Z. Mu, Y. Gong, W. Cai, and Y. Wang. Efficient local energy dissipation preserving algorithms for the Cahn–Hilliard equation. *Journal of Computational Physics*, 374:654–667, 2018.

[68] A. Nugent. Physical neural network design incorporating nanotechnology, May 3 2005. US Patent 6,889,216.

[69] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8024–8035, 2019.

[71] Y. Qiu and R. Suda. Train-by-reconnect: Decoupling locations of weights from their values. *Advances in Neural Information Processing Systems*, 33:20952–20964, 2020.

[72] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[73] W. Rudin. *Principles of mathematical analysis*. McGraw-Hill, New York, 3d edition, 1976.

[74] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[75] V. G. Satorras, E. Hoogeboom, and M. Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

[76] L. Scabini, B. De Baets, and O. M. Bruno. Improving deep neural network random initialization through neuronal rewiring. *arXiv preprint arXiv:2207.08148*, 2022.

[77] H. Sharma, M. Patil, and C. Woolsey. A review of structure-preserving numerical methods for engineering applications. *Computer Methods in Applied Mechanics and Engineering*, 366:113067, 2020.

[78] J. Shen, T. Tang, and L.-L. Wang. *Spectral methods: algorithms, analysis and applications*, volume 41. Springer Science & Business Media, 2011.

[79] J. Shen, J. Xu, and J. Yang. The scalar auxiliary variable (SAV) approach for gradient flows. *Journal of Computational Physics*, 353:407–416, 2018.

[80] Z. Shen, H. Yang, and S. Zhang. Optimal approximation rate of ReLU networks in terms of width and depth. *Journal de Mathématiques Pures et Appliquées*, 157:101–135, 2022.

[81] J. Song, W. Cao, F. Liao, and W. Zhang. VW-PINNs: A volume weighting method for PDE residuals in physics-informed neural networks. *Acta Mechanica Sinica*, 41(3):324140, 2025.

[82] E. M. Stein and R. Shakarchi. *Real analysis: measure theory, integration, and Hilbert spaces*. Princeton University Press, 2009.

[83] M. H. Stone. The generalized Weierstrass approximation theorem. *Mathematics Magazine*, 21(5):237–254, 1948.

[84] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27, 2014.

[85] M.-X. Wang and Y. Qu. Approximation capabilities of neural networks on unbounded domains. *Neural Networks*, 145:56–67, 2022.

[86] S. Wang, S. Sankaran, and P. Perdikaris. Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 421:116813, 2024.

[87] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[88] G. B. Whitham. *Linear and nonlinear waves*. John Wiley & Sons, 1999.

[89] W. L. Wood. An exact solution for Burger's equation. *Communications in Numerical Methods in Engineering*, 22(7):797–798, 2006.

[90] B. Yu and W. E. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[91] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[92] Y. Zang, G. Bao, X. Ye, and H. Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.

[93] G. Zeng, Y. Chen, B. Cui, and S. Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019.

[94] J. Zhang, S. Zhang, J. Shen, and G. Lin. Energy-dissipative evolutionary deep operator neural networks. *Journal of Computational Physics*, 498:112638, 2024.

[95] S. Zhang, Z. Shen, and H. Yang. Deep network approximation: Achieving arbitrary accuracy with fixed number of neurons. *Journal of Machine Learning Research*, 23(276):1–60, 2022.