Numerical Simulation of Semiconductor's Diffusion-Reaction Process Based on Deep Learning Methods

Master-Arbeit

zur Erlangung des Grades

Master of Science (M.Sc.)

im Studiengang Wirtschaftsmathematik

am Department Mathematik der
Friedrich-Alexander-Universität Erlangen-Nürnberg

vorgelegt am **20.January 2025**

von **Xuepeng Cheng**

Prüfer: Prof. Daniel Tenbrinck
Prüfer: Prof. Manuel Friedrich

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
NATURWISSENSCHAFTLICHE
FAKULTÄT

Department
MATHEMATIK

For my parents and girlfriend,
whose endless love and support have always been my foundation;
whose unwavering encouragement has inspired me every step of the way;
whose belief in me has given me the strength to overcome every challenge.

Acknowledgement

I would like to express my appreciate to Prof. Daniel Tenbrinck for his insightful advice and patient guidance throughout the journey of this thesis.

I would also like to thank the researchers at the Fraunhofer-Institute for all their helps within this thesis.

Contents

List of Figures

Chapter 1

Abstract

The aim of this study is to explore different architectures such as Physical Information Neural Network (PINN), PINN with the introduction of attention mechanism (Attention-PINN), PINN incorporating random Fourier features (Fourier-PINN), as well as KAN based on the B-spline function and ChebyKAN based on Chebyshev polynomials, for solving the system of diffusion reaction differential equations and compared accucacy and efficiency of these architectures. In order to provide a reliable benchmark, the finite difference method is employed to obtain the reference solution.

Through an exhaustive error analysis, we explore the differences in the causes of error generation between the multilayer perceptron (MLP) used in each PINN architecture and KAN and ChebyKAN, and propose corresponding improvements. Our research shows that although KAN still has advantages in certain scenarios(function fitting), MLP shows higher accuracy and computational efficiency when dealing with complex physics problems.

Keywords: PINN, KAN, Diffusion Reaction, Differential Equation

[0] https://github.com/Boroboroda

Chapter 2

Introduction

2.1 Background

In the modern era, semiconductors stand as a cornerstone technology underpinning nearly all electronic devices and systems. These materials, with their unique properties that can be precisely controlled and manipulated , have revolutionized various industries including telecommunications, energy, healthcare, automotive, computing and consumer electronics.

Semiconductors are elements or compounds with conductivity between that of conductors(like metals) and insulate(like glass). The most common semiconductor material is silicon, but others like gallium arsenide, germanium, and more recently, wide-bandgap material such as **silicon carbid (SiC)** and gallium nitride (GaN), play critical roles in specialized in specialized applications.

The importance of semiconductors cannot be overstated. They enable the miniaturization of electronic devices, leading to smaller, faster, and more efficient devices. Transistors, diodes, and integrated circuits (ICs) - all based on semiconductor technology - have allowed for the development of microprocessors that power everything from smartphones to supercomputers. Without them, we would not have experienced the digital revolution that trandsformed our world.

In the development of modern semiconductor devices, the study of the interaction between metal and semiconductor at interphase boundaries is of extremely important significance, especially in view of the complexity of forming rectifying and ohmic contacts. Especially for wide-bandgap semiconductors such as silicon carbid (SiC), fabricating ohmic contacts faces special difficulties. Chemical reactions between metal atoms and silicon carbide form either silicides or carbides. For instance, with metal Ni, Pd, Pt and Co, the product of the reaction is silicides, while with metal Ti, Ta, Cr and Zr, the product

is carbides. Here we consider the help of the most commonly used silicide-forming metal nickel to study its ohmic contact with $n-$SiC.

2.2 Model Between Ni and SiC

The simplest model of metal-semiconductor interactions involves interdiffusion from metal to semiconductor and from semiconductor to metal. Different atoms migrate in different directions, and the difference in their diffusivities leads to the Kirkendall effect[5].

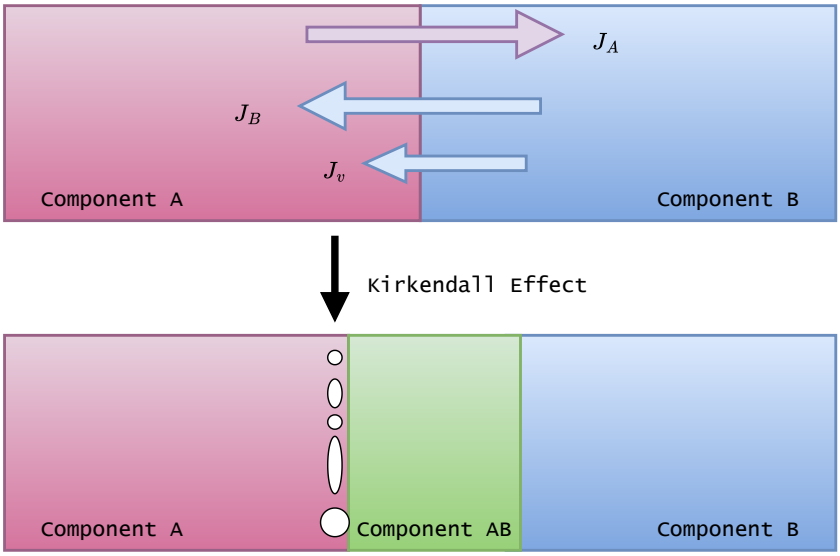


Figure 2.1: Kirkdoll Effect

In Darken's theory[6], two important phenomena are explained for the diffusion pair that forms a continuous solid solution: the mutual diffusion process and the displacement of interphase boundary. Due to the difference in the diffusivity of various atoms in a polyalloy, The plane of marked atoms perpendicular to the diffusion direction will be displaced along the diffusion direction. Assume that the diffusion system consists of two atoms, A and B. The marked atom plane moves at a rate v. At location x, the number of A atoms passing through unit area is:

$$J_A(x) = -D_A \frac{\partial n_A}{\partial x} + n_A v \tag{2.1}$$

where n_A is the number of A atoms per unit volume. At $x + dx$ the atomic flow of A is:

$$J_A(x+dx) = (-D_A \frac{\partial n_A}{\partial x} + n_A v) + \frac{\partial}{\partial x}(-D_A \frac{\partial n_A}{\partial x} + n_A)dx \tag{2.2}$$

So at x, the rate of change for the number of A atoms per unit volume is:

$$\frac{\partial n_A}{\partial t} = \frac{\partial}{\partial x}(D_A \frac{\partial n_A}{\partial x} - n_A v) \tag{2.3}$$

also,

$$\frac{\partial n_B}{\partial t} = \frac{\partial}{\partial x}(D_B \frac{\partial n_B}{\partial x} - n_B v) \tag{2.4}$$

In fact, in addition to A and B atoms, there should be vacancies in the unit volume, and their number is n_0. It can usually be expressed as $n_A + n_B + n_0 = C$, where C is a constant. Due to $n_0 \ll n_A + n_B$, commonly use $n_A + n_B = C$. From (2.3) and (2.4), we can obtain:

$$\frac{\partial}{\partial x}[D_A \frac{\partial n_A}{\partial x} + D_B \frac{\partial n_B}{\partial x} - (n_A + n_B)v] = 0, \tag{2.5}$$

$$\xrightarrow{\text{Solve equations}} D_A \frac{\partial n_A}{\partial x} + D_B \frac{\partial n_B}{\partial x} - (n_A + n_B)v = f(t) \tag{2.6}$$

At $x \to \infty$,

$$\begin{aligned} v = 0, \quad & \frac{\partial n_A}{\partial x} = \frac{\partial n_B}{\partial x} = 0 \to f(t) = 0 \\ v = & \frac{1}{n_A + n_B}(D_A \frac{\partial n_A}{\partial x} + D_B \frac{\partial n_B}{\partial x}) \end{aligned} \tag{2.7}$$

Plug these back into (2.3), it can be obtained:

$$\begin{aligned} \frac{\partial n_A}{\partial t} &= \frac{\partial}{\partial x}[D_A \frac{\partial n_A}{\partial x} - \frac{n_A}{n_A + n_B} D_A \frac{\partial n_A}{\partial x} - \frac{n_A}{n_A + n_B} D_B \frac{\partial n_B}{\partial x}] \\ &= \frac{\partial}{\partial x}[\frac{n_B D_A}{n_A + n_B} \frac{\partial n_A}{\partial x} - \frac{n_A D_B}{n_A + n_B} \frac{\partial n_B}{\partial x}], \\ &= \frac{\partial}{\partial x}[\frac{n_B D_A}{n_A + n_B} \frac{\partial n_A}{\partial x} + \frac{n_A}{n_A + n_B} D_B \frac{\partial n_A}{\partial x}], \end{aligned} \tag{2.8}$$

Reformulate it in the form of Fick's second law[7]:

$$\frac{\partial C_A}{\partial t} = \frac{\partial}{\partial x}(\tilde{D} \frac{\partial C_A}{\partial x}) \tag{2.9}$$

where $\tilde{D} = \frac{C_A D_B + C_B D_A}{C_A + C_B}$ is chemical diffusion coefficient.

In the proposed model from Aleksandrov[8], in the Ni-SiC pair, the metal interacts only with silicon and forming silicides. This chemical reaction is the main cause of the destruction of the rigid silicon-carbide lattice. In the nickel-silicon contact, the silicide formed is concentrated only at the phase interface. In the nickel-silicon carbide reaction, on the other hand, the reaction products are distributed in a relatively wide region that is comparable to the thickness of the initial metal film. The large range of reaction regions suggests a relatively low rate of solid-phase reaction compared to the diffusion rate of the mobile component.

In lieu of the epitaxial growth on the phase boundary, the silicide formation proceeds here by the so-called interstitial mechanism. In the process, fast diffusing metal interstitial atoms weaken and break neighboring covalent bonds in the semiconductor (in this case, Si-C bonds), subsequently replacing them with silicon-metal bonds.

At the same time, carbon is reduced to its elemental form and does not react with metals. Taking nickel as an example, the reaction equation to form metal monosilicide is as follows:

$$\mathrm{Ni} + \mathrm{SiC} \xrightarrow{k_{11}} \mathrm{NiSi} + \mathrm{C} \tag{2.10}$$

At relatively low annealing temperatures, the metal-enriched silicide(Ni_2Si) is formed:

$$\mathrm{Ni} + \mathrm{NiSi} \xrightarrow{k_{21}} \mathrm{Ni_2Si} \tag{2.11}$$

At higher temperatures, and under proton irradiation, it is nickel disilicide with liberation of carbon that is formed alongside with monosilicide in the reaction:

$$\mathrm{NiSi} + \mathrm{SiC} \xrightarrow{k_{12}} \mathrm{Ni_2Si} + \mathrm{C} \tag{2.12}$$

Here k_{11}, k_{21}, k_{12} are the rate constants for reactions (2.10), (2.11) and (2.12) respectively. Since the diffusivities of metal nickel and carbon have such relation that $D_{Ni} \gg D_C$. All these reactions proceed in the SiC bulk after the diffusion penetration of Ni into SiC. Due to the Kirkendall effect[5], there will be a displacement of interphase boundary to the surface.

According to Daken's theory[6], the mutal diffusion or the diffusion mixing of components in a binary system with infinite solubility is described by the effective heterogeneous diffusion coefficient(D^*). Assuming that the diffusion mixing of Ni, Si in SiC, and reduced C can also be represented by an effective heterogeneous diffusion coefficient (D^*) in the Ni-SiC system.

For this kinetic evolution system, reaction terms are introduced into the corresponding diffusion equations as the formation of new silicide phases. The corrected diffusion-reaction equation system is described as below:

$$\frac{\partial C_{\mathrm{Ni}}}{\partial t} = \frac{\partial}{\partial x}(D^* \frac{\partial C_{\mathrm{Ni}}}{\partial x}) - k_{11} C_{\mathrm{Ni}} C_{\mathrm{SiC}} - k_{21} C_{\mathrm{Ni}} C_{\mathrm{NiSi}}, \tag{2.13}$$

$$\frac{\partial C_{\mathrm{SiC}}}{\partial t} = \frac{\partial}{\partial x}(D^* \frac{\partial C_{\mathrm{SiC}}}{\partial x}) - k_{11} C_{\mathrm{Ni}} C_{\mathrm{SiC}} - k_{12} C_{\mathrm{NiSi}} C_{\mathrm{SiC}}, \tag{2.14}$$

$$\frac{\partial C_{\mathrm{C}}}{\partial t} = \frac{\partial}{\partial x}(D^* \frac{\partial C_{\mathrm{C}}}{\partial x}) + k_{11} C_{\mathrm{Ni}} C_{\mathrm{SiC}} + k_{12} C_{\mathrm{NiSi}} C_{\mathrm{SiC}}, \tag{2.15}$$

$$\frac{\partial C_{\mathrm{NiSi}}}{\partial t} = k_{11} C_{\mathrm{Ni}} C_{\mathrm{SiC}} - k_{21} C_{\mathrm{Ni}} C_{\mathrm{NiSi}} - k_{12} C_{\mathrm{NiSi}} C_{\mathrm{SiC}}, \tag{2.16}$$

$$\frac{\partial C_{\mathrm{NiSi_2}}}{\partial t} = k_{12} C_{\mathrm{NiSi}} C_{\mathrm{SiC}}, \tag{2.17}$$

$$\frac{\partial C_{\mathrm{Ni_2Si}}}{\partial t} = k_{21} C_{\mathrm{Ni}} C_{\mathrm{NiSi}}. \tag{2.18}$$

where t is the annealing time; x is the depth; and $C_i,\ i \in \{\text{Ni, SiC, C, NiSi, NiSi}_2\text{, Ni}_2\text{Si}\}$ are the concentrations of each components. The effective heterodiffusion coefficient has the following form:

$$D^* = D_{\text{Ni}} \frac{C_{\text{SiC}} + C_{\text{C}} + C_{\text{NiSi}} + C_{\text{NiSi}_2} + C_{\text{Ni}_2\text{Si}}}{C_{\text{Ni}} + C_{\text{SiC}} + C_{\text{C}} + C_{\text{NiSi}} + C_{\text{NiSi}_2} + C_{\text{Ni}_2\text{Si}}} \tag{2.19}$$

In equations (2.16), (2.17) and (2.18) there is no diffusion terms. This is due to the fact that in the steady state flow mode of the system, the plastic flow of the silicide material towards the outer surface, caused by the flux of nickel atoms, is compensated by the growth flow of the silicide phase in the opposite direction, namely Kirkendall Effect.

And the initial conditions of the system above are:

$$C_{\text{Ni}}(x,0) = \begin{cases} N_{\text{Ni}}, & 0 \le x \le h, \\ 0, & h < x \le L, \end{cases} \tag{2.20}$$

$$C_{\text{SiC}}(x,0) = \begin{cases} 0, & 0 \le x < h, \\ N_{\text{Si}}, & h \le x \le L, \end{cases} \tag{2.21}$$

$$C_i(x,0) = 0, \text{ for } i \in \{\text{C, NiSi, NiSi}_2\text{, Ni}_2\text{Si}\}. \tag{2.22}$$

where h is the metal-film thickness, L is the SiC-substrate thickness (the solution-region depth), N_{Ni} is the intrinsic metal atom concentration, and N_{Si} is the intrinsic silicon-atom concentration in the substrate. And the corresponding boundary conditions are:

$$\begin{gathered} \frac{\partial C_{\text{Ni}}}{\partial x}(x,t) = \frac{\partial C_{\text{SiC}}}{\partial x}(x,t) = \frac{\partial C_{\text{C}}}{\partial x}(x,t) = 0, \\ x = 0 \wedge x = L,\ 0 \le t \le T. \end{gathered} \tag{2.23}$$

2.3 Research purpose

The above model provides a good quantitative description of the interaction between Ni and single-crystal-line SiC, with special attention to mutual diffusion between components and volume silicide-formation reactions. In order to have an intuitive understanding of the solution to this system of differential equations, the traditional grid-based finite difference method (FDM) was first used as a reference solution, which has a proven application history and high accucacy when dealing wiht such physical and chemical processes.

On the one hand, the traditional finite difference method (FDM) is characterized by its straightforward and intuitive concept, ease of implementation, computational stability, and convergence properties. It boasts a well-established theoretical foundation and demonstrates high computational efficiency, particularly for problems with regular geometries and simple boundary conditions.

On the other hand, the accuracy of the finite difference method is heavily dependent on the quality of the mesh, with the generation of meshes becoming particularly challenging for intricate geometries. In the case of unstructured grids and complex boundaries, both the accuracy and efficiency of the calculations can be significantly compromised. Additionally, the handling of boundary conditions tends to be more intricate.

To address to limitations of these traditional numerical methods, deep learning (DL) provides an innovative paradigm for solving partial differential equations. Especially those neural networks that incorporate physical information (**PINNs**). As an emerging data-driven approach, by directly embedding physical constraints into the loss function, solutions to complex systems can be effectively learned by PINNs, relying on the ability to process complicate patterns and nonlinear relationships of deep neural networks (DNN) to approximate solutions to PDEs. It is because of "embedding" physical constraints, the predictions will progressively conform to known physical laws, and learn the structure of solutions from limited amount of data.

Compare to the multi-layer-perceptron (MLP) commonly used in PINNs, Kolmogorov-Arnold Networks (KAN) is seen as a strong contender, KAN not only inherits MLP's ability to handle nonlinear relationships, but also improves the model's expression ability and parameter efficiency by introducing features such as learnable activation functions and spline interpolation. Specifically, MLP has fixed activation function and learnable weight w and bias b; while KAN has learnable activation function on "edges". The network is able to adaptively adjust the nonlinear transformation of each connection/edge to more accurately capture complex patterns in the data. Compared with MLP of the same scale, it can achieve better fitting results with fewer parameters.

However, nothing in the world is perfect. Training process of neural network is often computationally intensive and extremely sensitive to the choice of network architecture and hyperparameters. The convergence and generalization ability of the network is affected by many factors, such as the accurate representation of the physical constraints, the design of the loss function, and the quality of the training data.

Therefore, this study aims to comprehensively compare traditional method and deep learning methods from multiple perspectives, including computational accuracy, computational efficiency, the impact of different hyperparameters on computational results, and performance across various problem types. Through an in-depth comparative analysis, we seek to provide more nuanced insights into the numerical simulation of complex physical problems and explore the complementary potential of these methodological approaches.

2.4 Related Works

In the field of solving PDEs, deep learning has been developed to move away from the reliance on manual derivation of analytic solution and discretized grids, provided efficient approximations for real-world scenarios by learning the PDE's solutions directly from

data. PINNs[9] is an application of deep learning methods in physical problems. The version proposed by Raissi is one of the earliest and most well-known implementations in this field and has important reference value.

Although it has achieved remarkable success in solving simple PDEs and specific physics tasks, PINNs still faces several important challenges when it comes to complex problems. These challenges include limited understanding of the underlying principles, sparse experimental data, and difficulties in accurate modeling. Moreover, these data-driven models, often characterized as "black box" approaches, may struggle to unveil the underlying mathematical structure of partial differential equation (PDE) solutions and may not be able to systematically incorporate domain-specific physical knowledge.

In recent years, many researchers have made various improvements based on the original architecture, including optimizing model construction, improving convergence speed, and developing software libraries.

2.4.1 Model Architecture Design

The current research scope of PINNs has expanded to include not only second-order linear PDEs, but also higher-order equations, fractional-order equations, and nonlinear partial differential equations (NLPDEs). Recently, researchers have conducted in-depth studies on the network structure of PINNs and developed various versions of PINNs specialized for different types of PDEs. For example, Pang et al. proposed fPINNs[10], which are suitable for integral equations and fractional-order PDEs; in the fields of energy, economics, and medicine, where high-order nonlinear partial differential equations with significant application value dominate, Bai et al. developed dPIRBNs[11] that can better handle NLPDEs with high-frequency characteristics and ill-posed computational domains.

Moreover, emerging research increasingly highlights the potential of integrating transfer learning[12, 13] and meta-learning[14, 15] paradigms with Physics-Informed Neural Networks (PINNs) to address complex time-dependent and high-dimensional partial differential equations (PDEs). These advanced methodological approaches enable more adaptive and efficient learning strategies by leveraging pre-trained knowledge from related problem domains and subsequently refining model performance through targeted fine-tuning.

As the potential of PINN is explored, researchers are increasingly aware of the challenges of PINN in dealing with large-scale domains or high-dimensional non-convex optimization problems. These problems are usually accompanied by higher absolute errors, higher training costs, and longer training times. Furthermore, when solving discrete solutions such as compressed Euler equations, the technique may have difficulty handling discontinuous solutions (such as shock waves and contact surfaces) if the initial conditions are not sufficiently smooth.

To overcome these obstacles, many experts adopt the principle of Domain Decomposition. This method divides the computational domain into discrete subdomains to achieve a divide and conquer strategy. By using PINN independently in each subdomain, parallel computations can be performed simultaneously, significantly reducing computational costs and achieving satisfactory results. Meng et al. proposed the pPINN method[16], which uses coarse-grained decomposition to decompose the long time domain into multiple discrete short time segments for parallel solution. Kharazmi et al. proposed hp-VPINNs[17] based on subgrid Galyokin's method to handle singularities and steep solutions efficiently.

2.4.2 Convergence Speed Optimization

PINNs typically require extensive training data and a complex optimization process to achieve a relatively good performance. Increasing the convergence speed can significantly reduce the training time, lower the training cost, and improve the usefulness of the model. In addition, faster convergence speed can enhance the model's ability to achieve global optimality, thereby improving its robustness and stability.

In PINNs, the constructed loss function is mainly minimized by adjusting the neural network parameters using optimization algorithms. Therefore, a correct selection of optimization algorithm can help avoid problems such as local minima and slow convergence that are common in large samples and deep networks. In terms of learning rate optimization, the ADAM optimizer is widely used due to its adaptive nature. Researchers at Marshall University use the ADAM optimizer to solve viscous and thermal boundary layer problems[18]. Some researchers combine L-BFGS and ADAM optimizers together and reveal that better results are obtained with ADAM + L-BFGS. And an improved Newton method (NysNewton-CG)[19] with damping is also proposed to avoid the results from falling into a local minimum.

The loss function of PINNs is constructed from the residuals of governing equations, initial/boundary conditions, and experimental data (optional). For a dynamical evolution system, it is essential to emphasize the importance of the initial conditions. So giving it a higher weight is an intuitve choice. However, inappropriate settings may have an impact on the model and impair the convergence process. An adaptive loss balancing method (lbPINNs)[20] based on maximum likelihood estimation was proposed by Xiang et al. Based on the characteristic that the residual term of the governing equations is zero, Yu et al. proposed Gradient-enhanced PINNs (gPINNs)[21] to improve the overall solution accuracy within the framework.

2.4.3 Development of Software Libraries

The training process of popular deep learning frameworks in Python such as Tensorflow, Pytorch, Jax are commonly used to train a PINNs model. These frameworks provide a

rich set of features and tools that enable researchers to easily build, train, and deploy different types of physical models.

Professor Karniadakis' team in the Department of Applied Mathematics at Brown University developed the DeepXDE[22] library for the scientific and educational communities. For beginners, the library provides an end-to-end solution and a large number of demonstration use cases.

2.5 Our Contribution

In this research, we work on several challenges faced by neural networks when applied to physical problems and propose corresponding solutions. The following are the main contributions of our work:

1. **Introduction of Nondimensionalization:** In view of the high sensitivity of neural networks to the scale of input data, which may lead to unstable or inefficient training, we introduced a nondimensionalization method. By normalizing all input features into the $[0, 1]$ interval, it ensures that the relative gap between different features is minimized, thus improving the generalization ability and training stability of the model. This data preprocessing strategy lays the foundation for subsequent model optimization.

2. **Random Fourier Features Embedding:** In order to enhance the expressive ability of the model and reduce the spectral bias, we introduce random Fourier feature embedding. This method maps the original input into a high-dimensional space through a kernel function, allowing the model to better capture complex patterns in the input data.

3. **Improved PINN and Attention Mechanism:** Inspired by the successful application of the Attention mechanism in the Transformer architecture, we adopted an improved PINNs architecture. This architecture innovatively introduces two encoders in the form of a moving average to build a new PINN similar to the Transformer structure. This improvement not only effectively alleviates the problems of gradient explosion and disappearance, but also significantly improves the accuracy of model prediction, making it excellent in solving complex physical problems.

4. **Non-negative Constraint Layer:** In practical application scenarios, it is an important physical fact that the concentration value always remains non-negative. In order to make the model output closer to the real situation, we added a forward constraint layer before the output stage of the neural network to force the output result to remain positive. This ensures the physical rationality of the prediction results and avoids the occurrence of unreasonable negative values.

5. **KAN Model:** We not only implemented the KAN network with B-spline as the basis function of the learnable activation function, but also developed the ChebyKAN

network with Chebyshev polynomial (Cheby-Poly) as the basis function. Through comparative experiments between these two models and MLP-based PINN and IA-PINN, their respective advantages and applicable scenarios were verified, providing new ideas and directions for future research.

6. **Error Theory:** We deeply analyze the differences between traditional MLP and KAN networks at the error level and explore the reasons for these differences. Through systematic comparisons, we reveal the advantages and disadvantages of different network structures in dealing with specific types of physical problems, providing a theoretical basis for further optimizing the model performance.

Chapter 3

Theoretical Foundations

3.1 Finite Difference Method

In numerical analysis, the finite-difference methods (FDM) are a class of numerical techniques for solving differential equations by approximating derivatives with finite differences. Both the spatial domain and temporal domain (if applicable) are discretized, or be seperated into a finite number of intervals, and the values of solution at the end points of the intervals are approximated by solving algebraic equations containing finite differences and valuse from nearby points.

Finite difference methods convert ordinary differential equations (ODE) or partial differential equations (PDE), which may be nonlinear, into a system of linear equations that can be solved by matrix algebar techniques. Modern computers can perform these linear algebra computations efficiently, and this, along with their relative ease of implementation, has led to the widespread use of FDM in modern numerical analysis[1]. Today, FDMs are one of the most common approaches to the numerical solution of PDE, along with finite element methods.

3.1.1 From Taylor Expansion to Difference

For a n-times differential function, by Taylor's Theorem the Taylor series expansion is given as:

$$f(x_0+h) = f(x_0) + \frac{f^{(1)}(x_0)}{1!}h + \frac{f^{(2)}(x_0)}{2!}h^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n + \mathrm{R}_n(x), \tag{3.1}$$

By truncating the Taylor polynomial expansion part and the remainder part, we can derive the first order derivative of the function f.

$$f(x_0+h) = f(x_0) + f^{(1)}(x_0)h + R_1(x) \longrightarrow \frac{f(x_0+h)}{h} = \frac{f(x_0)}{h} + f^{(1)}(x_0) + \frac{R_1(x)}{h} \tag{3.2}$$

Assuming that the remainder part is sufficiently small, then by organizing (3.2), we can obtain an approximation of the first-order derivative of f.

$$f^{(1)}(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h} \tag{3.3}$$

This is formally similar to the definition of the f derivative, which is:

$$f^{(1)}(x_0) = \lim_{h\to 0} \frac{f(x_0+h)-f(x_0)}{h} \tag{3.4}$$

except for the limit towards zero (the method is named after this).

3.1.2 Difference Method

Consider taking a simple 1-dimensional normalized heat equation with homogeneous Dirichlet boundary condition as an example:

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \\ u(0,t) = u(1,t) = 0, \\ u(x,0) = u_0(x). \end{cases} \tag{3.5}$$

First partition the spatial and temporal domain into $\{x_0, \cdots, x_J\}$ and $\{t_0, \cdots, t_N\}$, respectively. The meshing is assumed to be uniform, so that the difference between two spatially continuous points is directly h, and the difference between two temporally continuous points is k. At point (x_j, t_n) obtained an approximate numerical solution repreented as $u(x_j, t_n) = u_j^n$.

At time t_n and location x_j, a second-order central difference for spatial term (forward time-centered space i.e. FTCS) has the recurrence equation:

$$\frac{u_j^{n+1}-u_j^n}{k\Delta t} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2} \tag{3.6}$$

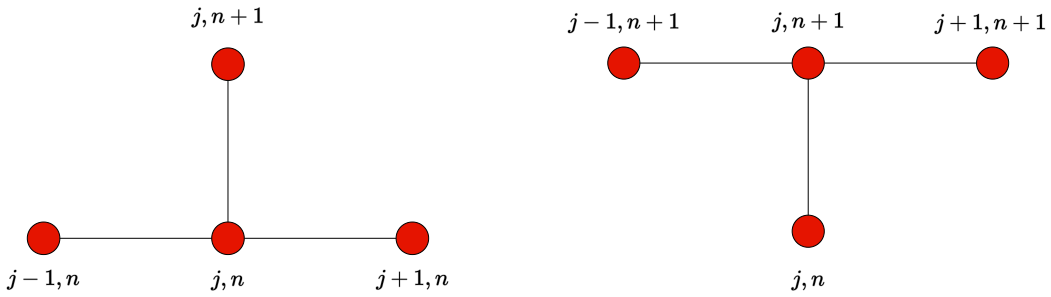
This is an explicit method. In the boundary and initial parts, just replace with the given conditions. And u_j^{n+1} can be described as:

$$u_j^{n+1} = (1-2r)u_j^n + ru_{j-1}^n + ru_{j+1}^n, \quad \text{where} \quad r = k\frac{\Delta t}{h^2} \tag{3.7}$$

According to this recurrence relation, by knowing the value at moment t_n, the corresponding value at moment t_{n+1} can be determined. This explicit method is known to be numerically stable and convergent whenever $r \leq \frac{1}{2}$[2]. The numerical errors are proportional to the time step and the square of the space step with $\Delta u = O(k) + O(h^2)$.

At time t_{n+1} and location x_j, a second-order central difference for spatial term (backward time-centered space i.e. BTCS) gives the recurrence equation:

$$\frac{u_j^{n+1}-u_j^n}{k\Delta t} = \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h^2} \tag{3.8}$$



(a) The explicit method stencil.

(b) The implicit method stencil.

Figure 3.1: Schematic diagram of the finite difference method.

This is an implicit method. And u_j^{n+1} can be obtained from:

$$u_j^n = (1+2r)u_j^{n+1} - ru_{j-1}^{n+1} - ru_{j+1}^{n+1} \tag{3.9}$$

The scheme is always numerically stable and convergent but usually more numerically intensive than the explicit method as it requires solving a system of numerical equations on each time step. The errors are linear over the time step and quadratic over the space step with $\Delta u = O(k) + O(h^2)$.

When solving heat equations and partial differential equations of similar forms, the Crank-Nicolson[23] method is widely used. It can be shown that the C-N method is unconditionally stable for the diffusion equation (as well as many other equations)[3].

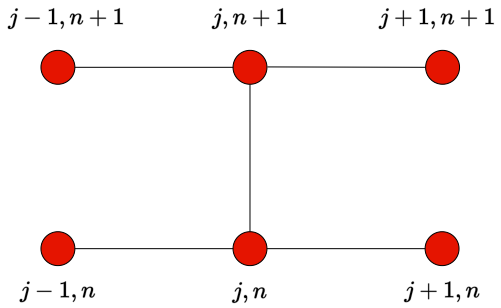


Figure 3.2: The CrankNicolson stencil.

The C-N method uses central difference in the spatial domain; the trapezoidal formula is applied in the temporal domain to ensure second-order convergence. The difference equation here uses the average of the forward Euler method and the backward Euler method, which is implicitly dependent on the solution and is not a simple mean value.

$$\frac{u_j^{n+1}-u_j^n}{k\Delta t}=\frac{1}{2}(\frac{u_{j+1}^n-2u_j^n+u_{j-1}^n}{h^2}+\frac{u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1}}{h^2}) \tag{3.10}$$

So applying the above methods to our system of diffusion reaction equations and take Nickel (2.13) for instance, we have:

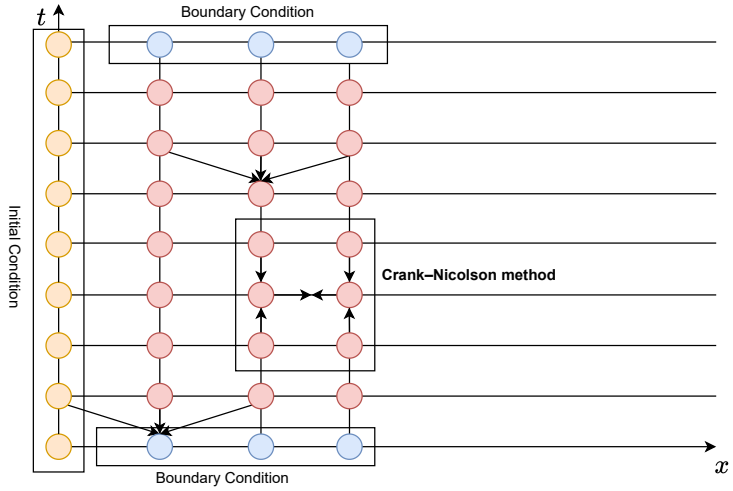


Figure 3.3: Diffusion Reaction FDM discretized grid.

Discretization of the time derivative:

$$\frac{\partial C_{\mathrm{Ni}}}{\partial t}\approx\frac{C_{\mathrm{Ni}}^{n+1}-C_{\mathrm{Ni}}^n}{\Delta t}$$

Discretization of spatial derivatives:

$$\frac{\partial}{\partial x}(D^*\frac{\partial C_{\mathrm{Ni}}}{\partial x})\approx\frac{1}{\Delta x}[D_{i+\frac{1}{2}}\frac{C_{\mathrm{Ni}(i+1)}^n-C_{\mathrm{Ni}(i)}^n}{\Delta x}-D_{i-\frac{1}{2}}\frac{C_{\mathrm{Ni}(i)}^n-C_{\mathrm{Ni}(i-1)}^n}{\Delta x}]$$

where we can caculate $D_{i+\frac{1}{2}}=\frac{D_i+D_{i+1}}{2}$ and $D_{i-\frac{1}{2}}=\frac{D_i+D_{i-1}}{2}$.

$$\begin{aligned}&-k_{11}C_{\mathrm{Ni}}C_{\mathrm{SiC}}-k_{21}C_{\mathrm{Ni}}C_{\mathrm{NiSi}}\approx\\&-k_{11}\frac{1}{2}(C_{\mathrm{Ni}}^{n+1}+C_{\mathrm{Ni}}^n)\frac{1}{2}(C_{\mathrm{SiC}}^{n+1}+C_{\mathrm{SiC}}^n)\\&-k_{21}\frac{1}{2}(C_{\mathrm{Ni}}^{n+1}+C_{\mathrm{Ni}}^n)\frac{1}{2}(C_{\mathrm{NiSi}}^{n+1}+C_{\mathrm{NiSi}}^n)\end{aligned}$$

At the given boundary $\frac{\partial C}{\partial x}$ with Neumann condition, by the first-order realization of the temporal-like form above.

3.2 Deep Learning Method

Physics-Informed Neural Networks (PINNs), as its name suggests, is a revolutionary hybrid modeling paradigm that cleverly combines the powerful representational capabilities of deep learning with the fundamental laws of the physics. An intuitive representation of PINNs is that it consists of two parts, the neural network (NN) part and the physical information (PI) part.

By leveraging the neural network's output, we construct loss functions that inherently incorporate the a priori physical constraints. Through a carefully designed optimization framework that minimizes residuals, we iteratively refine the network's parameters, enabling the model to progressively converge towards the true solution of the physical system.

3.2.1 Original PINN framework

In the reference [9], original PINN framework is proposed to solve partial differential equations using deep neural networks. The loss function consists of residual terms and supervision terms. The degree of approximation of deterministic conditions such as initial conditions and boundary conditions is measured by the supervision term; the residual term is obtained through automatic differentiation techniques, and thus the degree of approximation of the governing equations is measured. This framework has a wide range of applicability and powerful general capabilities. From hydrodynamic equations to heat transfer equations, from electromagnetic equations to diffusion reaction equations, it is applicable to almost most scenarios.

Here we give a generalized nonlinear differential equation:

$$\begin{cases} \mathcal{D}(u(x,t)) = f(x,t), & x \in \Omega \subset \mathbb{R}^d, \\ \mathcal{B}(u(x,t)) = g(x,t), & x \in \partial\Omega, \\ \mathcal{I}(u(x,0)) = h(x), & x \in \Omega. \end{cases} \tag{3.11}$$

where $\mathcal{D}$ denotes the differential operator; $\mathcal{B}$ is the boundary condition operator; and $\mathcal{I}$ denotes the initial condition operator. And $\mathcal{B}$ is usually a Dirichlet, Newman, Robin, periodic or mixing condition. $u(x,t)$ is the solution function on the d-dimensional domain Ω, $f(x,t)$ is a given source or governing function that introduces priori influences into the system.

In Physics-Informed Neural Networks (PINNs), the loss function is meticulously crafted to integrate the neural network solution's fidelity across initial conditions, boundary conditions, and the underlying partial differential equation (PDE). To harmonize the

contributions of these diverse constraints, we introduce weighted coefficients λ_r, λ_{bc}, and λ_{ic}, corresponding to initial conditions, boundary conditions, and PDE residual terms, respectively. Through strategic calibration of these weighting parameters, we can dynamically modulate the relative importance of each physical constraint during the model training process, thereby enhancing the solution's accuracy, physical interpretability, and overall numerical stability.

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_{bc} \mathcal{L}_{bc} + \lambda_{ic} \mathcal{L}_{ic} \tag{3.12}$$

where $\mathcal{L}_r$ denotes the loss term for residuals corresponding to the PDE; $\mathcal{L}_{bc}$ denotes the loss term for boundary conditions; and $\mathcal{L}_{ic}$ for initial conditions.

These loss terms are constructed in the form of mean squared error (MSE) below:

$$\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} |\mathcal{D}(\hat{u}(x_r^i, t_r^i) - f(x_r^i, t_r^i))|^2, \tag{3.13}$$

$$\mathcal{L}_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |\mathcal{B}(\hat{u}(x_{bc}^i, t_{bc}^i) - g(x_{bc}^i, t_{bc}^i))|^2, \tag{3.14}$$

$$\mathcal{L}_{ic} = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |\mathcal{I}(\hat{u}(x_{ic}^i, 0) - g(x_{ic}^i, 0))|^2. \tag{3.15}$$

where $\hat{u}(x, t; \theta)$ is the neural network prediction; (x, t) is the spatio-temporalcoordinates; N_r, N_{bc}, N_{ic} are the collocation of sampled points generated according to domain Ω and boundary/initial conditions. Later a gradient descent based optimization algorithm is used to minimize this loss function(3.12). Once the loss $\mathcal{L}$ reach the given threshold as close to zero as possible, we can regard the neural network $\hat{u}(x, t; \theta)$ as a good enough approximation to function $u(x, t)$ of (3.11).

3.2.2 Improved Multilayer Perceptron

The gradient vanishing/explosion problem encountered by oringinal multilayer perceptrons when the network structure is too deep or when dealing with complex problems is a challenge that PINNs are prone to fall into during the training process. A recently improved fully connected feedforward neural network[24], inspired by the fruitful attention mechanism in Transformer[25], successfully alleviated this dilemma.

By using two different encoders U and V in the feature space, this improved structure enhances the expressive ability of PINN by embedding the input variable $\boldsymbol{x}$ into the hidden state.

$$U = \sigma(\boldsymbol{x}_0 W^U + b^U), \quad V = \sigma(\boldsymbol{x}_0 W^V + b^V) \tag{3.16}$$

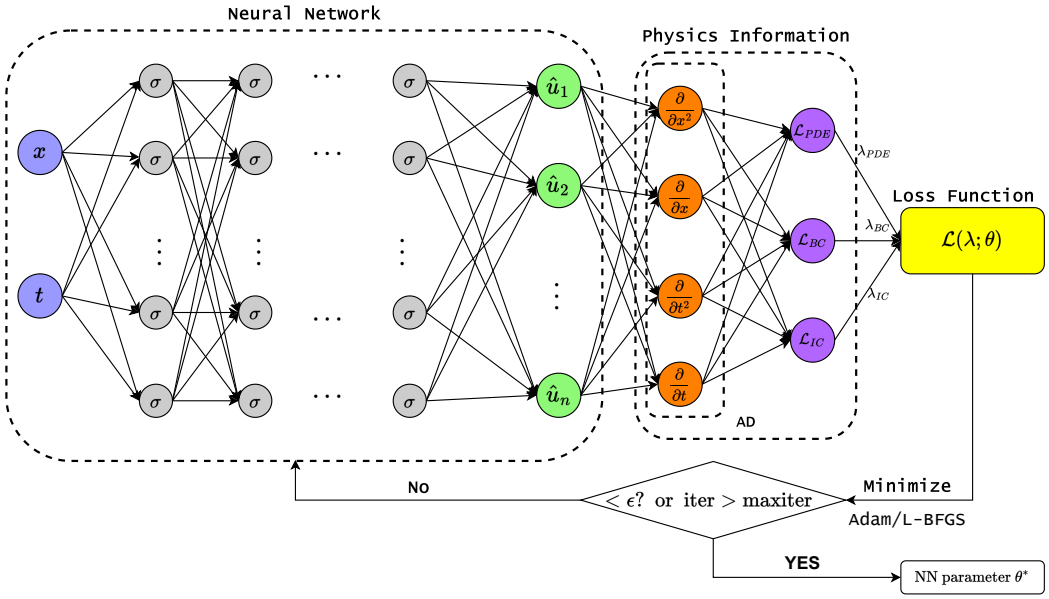


Figure 3.4: PINN Framework

The encoder is then assimilated into each hidden layer of the conventional MLP by point-by-point multiplication. Thus, each forward propagation has the form of:

$$\alpha^l(\boldsymbol{x}) = \alpha^{l-1}(\boldsymbol{x})W^l + b^l, \quad \text{for } \ l \in \{1, 2, \cdots, N\}. \tag{3.17}$$

$$\alpha^l(\boldsymbol{x}) = \sigma(\alpha^l(\boldsymbol{x})), \tag{3.18}$$

$$\alpha^l(\boldsymbol{x}) = \alpha^l(\boldsymbol{x}) \odot U + (1 - \alpha^l(\boldsymbol{x})) \odot V, \tag{3.19}$$

where $\boldsymbol{x}$ is the input, α^l and W^l are the neurons and weights of the $l-$th layer, σ is the activation function, and $\odot$ is the element-by-element product.

3.2.3 Kolmogorov Arnold Networks

Inspired by the Kolmogorov-Arnold representation theorem[26], Liu et al. proposed Kolmogorov Arnold Networks[27] (KANs) as promising alternatives to Multi-Layer Perceptrons (MLPs). MLP, we mentioned in last section, has fixed activation functions on nodes ("neurons") in the form of (3.14) and learnable parameters including weights W and bias b (3.13). However, one obvious difference is that KANs have learnable activation functions on edges. That is, KAN does not perform linear transformation by weights and biases on the edge - in lieu of every weight parameter, a univariate function parametrized as a spline was introduced.

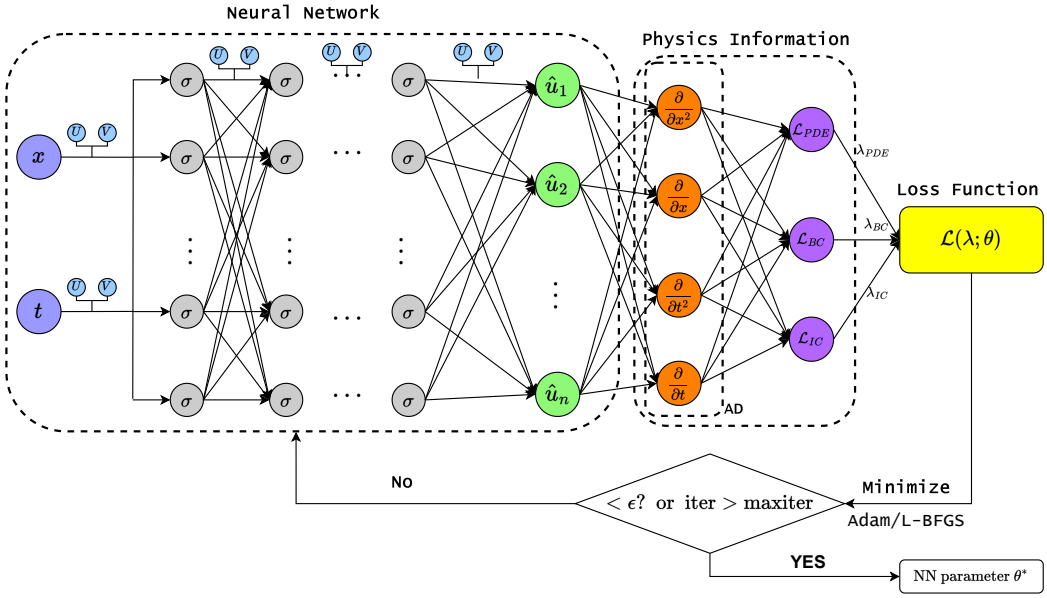


Figure 3.5: IA-PINN Framework

Vladimir Arnold and Andrey Kolmogorov on the basis of solving Hilbert's 13-th problem, the following theorem is proved using elementary methods.

Theorem 3.1 (KolmogorovArnold Representation Theorem)**.** *For $n \geq 2$, there exists continuous functions $\psi^{p,q}(x), p = 1, \cdots, n;\ q = 1, \cdots, 2n+1$ on interval $I = [0,1]$ that satisfy, for any continuous function $f(x_1, \cdots, x_n) \in C^0(I^n)$, there are $2n+1$ continuous function $\varphi_q(y)$ of one variable, such that,*

$$f(x_1, \cdots, x_n) = \sum_{q=1}^{2n+1} \varphi_q(\sum_{p=1}^{n} \psi^{p,q}(x_p)) \tag{3.20}$$

In a sense, they showed that the only true multivariate function is addition. This means that, theoretically, complex multivariate functions can be decomposed into combinations of simple univariate functions. However, these univariate functions may be non-smooth or even fractal, so they may be difficult to implement in practice. Vitushkin[26] proved that the inner function $\psi^{p,q}$ of KA-Theorem has poor smoothness, even not C^1. And Sprecher gave out a constructive proof[28].

Most functions in science and daily life are often smooth and have sparse compositional structures, Therefore it is important to find a smoothed "Kolmogorov-Arnold representation". When KAN layer with n_{in}-dimensional inputs and $n_{out}-$dimensional outputs,

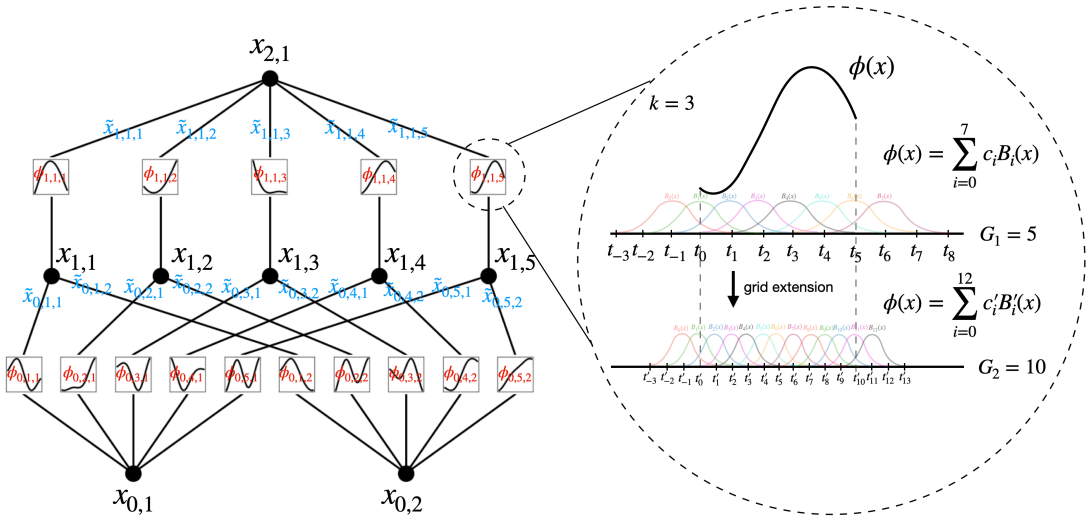


Figure 3.6: Left: Notations of activations that flow through the network. Right: an activation function is parameterized as a B-spline, which allows switching between coarse-grained and fine-grained grids.[27]

it can be defined as a matrix of univariate functions.

$$\Psi = \{\psi_{q,p}\}, \qquad p = 1, 2, \cdots, n_{in}; \qquad q = 1, 2, \cdots, n_{out}. \tag{3.21}$$

In the theorem 3.1, the inner functions form a KAN layer with $n_{in} = n$ and $n_{out} = 2n+1$, and the outer functions forma KAN layer with $n_{in} = 2n+1$ and $n_{out} = 1$. Thus according to (3.20), Kolmogorov-Arnoldrepresentations simply consist of two KAN-Layers.

To stack $(L \gg 2)$ -KAN layers, shape of such KAN is represented by an integer list $[n_0, n_1, \cdots, n_L]$, where n_i is the number of nodes in the $i-$th layer. The i^{th} neuron in the l^{th} layer is defined as (l, j), and the activation value of (l, j)-neuron is $x_{l,i}$. It's easy to know between l^{th} and $l + 1^{th}$ layer there are $n_l \cdot n_{l+1}$ activations, and the activation function connects (l, i) and $(l + 1, i)$ is donated by:

$$\psi_{l,j,i}, \quad l = 0, \cdots, L-1; \quad i = 1, 2, \cdots, n_l; \quad j = 1, \cdots, n_{l+1}. \tag{3.22}$$

The pre-activation and post-activation of $\psi_{l,j,i}$ are $x_{l,i}$ and $\tilde{x}_{l,j,i}$ respectively. Unlike MLP, which needs to use a nonlinear activation function to "activate" the linear transformation results of edges on the neurons, the work of the KAN neurons is very simple, that is, summing up the results received by the $(l + 1, j)$ neurons.

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \psi_{l,j,i}(x_{l,i}). \tag{3.23}$$

In matrix form, it can be written as:

$$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \psi_{l,1,1}(\cdot) & \psi_{l,1,2}(\cdot) & \cdots & \psi_{l,1,n_l}(\cdot) \\ \psi_{l,2,1}(\cdot) & \psi_{l,2,2}(\cdot) & \cdots & \psi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{l,n_{l+1},1}(\cdot) & \psi_{l,n_{l+1},2}(\cdot) & \cdots & \psi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\mathbf{\Psi}_l} \mathbf{x}_l, \tag{3.24}$$

Assume $f(x)(x \in \mathbb{R}^n)$ has representation:

Definition 3.1.

$$KAN(x) = (\Psi_{L-1} \circ \Psi_{L-2} \circ \cdots \Psi_1 \circ \Psi_0)(x), \tag{3.25}$$

And it can be rewrite in the form of (3.20), assuming output dimension $n_L = 1$, and define $f(x) \equiv KAN(x)$:

$$f(\mathbf{x}) = \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{L-1,i_{L-1}} \left(\sum_{i_{L-2}=1}^{n_{L-2}} \cdots \left(\sum_{i_2=1}^{n_2} \phi_{2,i_3,i_2} \left(\sum_{i_1=1}^{n_1} \phi_{1,i_2,i_1} \left(\sum_{i_0=1}^{n_0} \phi_{0,i_1,i_0}(x_{i_0}) \right) \right) \right) \cdots \right), \tag{3.26}$$

Inspired by the residual connection in ResNet[29], the activation function $\psi(x)$ is expressed as the sum of a basis function $b(x)$ and a spline function:

$$\psi(x) = w(b(x) + \text{spline}(x)), \tag{3.27}$$

$$b(x) = silu(x) = \frac{x}{1 + e^{-x}}. \tag{3.28}$$

Spline function is parametrized as a linear combination of B-splines such that

$$\text{spline}(x) = \sum_i c_i B_i(x), \tag{3.29}$$

where c_i is trainable.

In order to quantitatively analyze the complexity of KAN, for a KAN with a depth of L, a width of N, a spline order of $k = 3$, and a grid number of G, its number of parameters is $O(N^2L(G+k))$ $O(N^2LG)$. On the contrary, for an MLP of the same size, the number of parameters is $O(N^2L)$.

Although small change like (3.27) have given vitality to the application of KAN on computers, errors have also appeared accordingly. I will discuss the advantages and disadvantages of analyzing KAN and MLP from error theory in subsequent chapters [4].

3.2.4 Chebyshev Polynomial Kolmogorov Arnold Networks

In the field of approximation theory[30, 31], Chebyshev polynomials play a very important role. These orthogonal polynomials, defined on [-1,1], have many unique properties, and here are some keys to Chebyshev polynomials:

$$T_0(x) = 1, \tag{3.30}$$
$$T_1(x) = x, \tag{3.31}$$
$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad n \geq 2 \tag{3.32}$$
$$T_n(x) = \cos(n \arccos(x)), \quad x \in [-1, 1]. \tag{3.33}$$

Several of its desirable properties make Chebyshev polynomials well suited for function

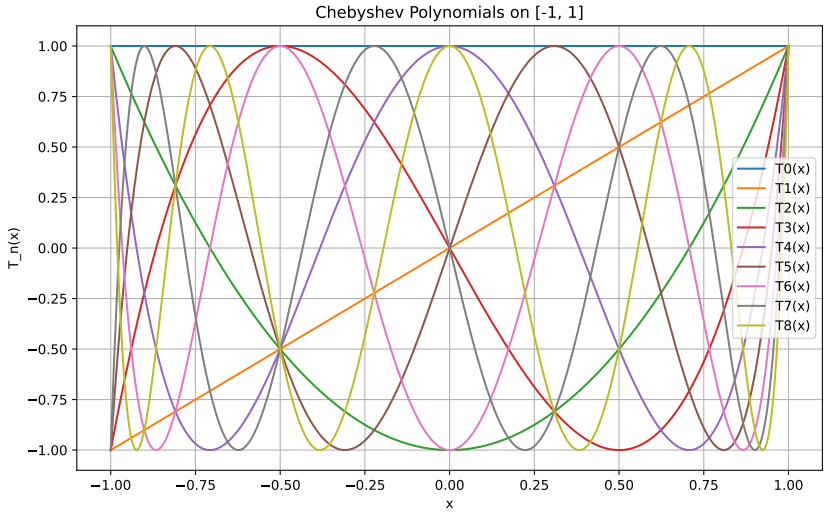


Figure 3.7: Plot of the first 10 T_n Chebyshev polynomials

approximation:

1. **Min-Max Property:** For any given $n \geq 1$, among the polynomials of degree n with leading coefficient 1, $p(x) = \frac{1}{2^{n-1}}T_n(x)$ is the one of which the maximal absolute value on the interval [-1, 1] is minimal. It means that the Chebyshev polynomial is the "most uniform" polynomial of a given degree.

2. **Orthogonality:** The sequence of Chebyshev polynomials is relative to the weight function $w(x) = (1 - x^2)^{\frac{1}{2}}$ orthogonal polynomials on the interval [-1, 1]. This means that for any two Chebyshev polynomials $T_m(x)$ and $T_n(x)$ of different degrees, their inner product is zero. This orthogonality simplifies the computation of coefficients in polynomial expansions and helps to reduce the accumulation of errors in numerical computations.

3. **Convergence speed:** The approximation error of a continuous function using Chebyshev polynomials decreases rapidly as the degree of the polynomials in-

creases, typically faster than other polynomial bases. The oscillatory behavior of the Chebyshev polynomials effectively distributes the errors, reducing the large errors that can occur near the endpoints.

4. **Easy To Implement:** Chebyshev polynomials can be easily computed by simple recurrence relations. The coefficients of such polynomials are usually small, so the recursive formulas are relatively stable and better numerical stability is generally obtained.

From Chebyshev polynomials to Cheby-KAN:

We introduce the Chebyshev polynomials in the form of a Komogrov-Arnold representation theorem based on (3.20). Replace the spline function in (3.27) with Chebyshev polynomials. However, it is important to note here that a hyperbolic tangent transformation of the input variable x is required making the input $x \to \tanh(x) \in [-1, 1]$. In this way layer normalization is necessary to effectively avoid the problem of vanishing gradients caused by hyperbolic tangent functions.

This formulation directly approximates the objective function $f(x)$ as a weighted sum of Chebyshev polynomials. It leveraging the Kolmogorov-Arnold Theorem's guarantee of the existence of a superposition of univariate functions to represent any continuous multivariate function.

Chapter 4

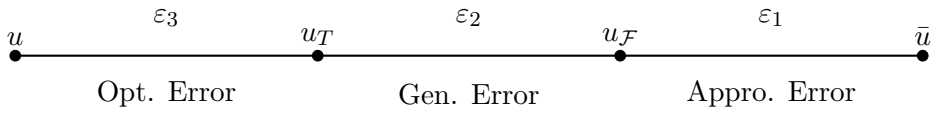
Perspective on Error Theory

After the account of some theoretical foundations in the previous chapter, a question naturally arises, how to analyze how good the approximations of MLP and KAN are? From errors provides a unique perspective.

Assume that we want to fit the true solution $\bar{u}(x)$ with network, then some components are essential.

- **Function Model**: $u(x;\theta)$, θ is the parameter that need to be optimized, and $x \in \Omega$.
- **Sample Points**: We may only know the grounded solution on some specific points $\{(x_i, \bar{u}_i)\}$.
- **Loss Function**: Measure the difference between model output and true solution: $\mathcal{L}(\theta) = \text{MSE}(u(x;\theta), \bar{u}(x))$.
- **Optimal Approach**: Obtain the optimal parameters θ^* by minimizing Loss $\mathcal{L}(\theta)$

Then errors are inevitable in this computational numerical simulation workflow. Next I will analyze the reasons for the emergence of errors in MPL and KAN at three levels.



4.1 Approximation Error

Let $\mathcal{F}$ be the set of functions in the form of $u(\cdot;\theta)$. Since true solution $\bar{u}$ may not exactly in $\mathcal{F}$, so to find a function $u_{\mathcal{F}} \in \mathcal{F}$ such that $u_{\mathcal{F}}$ is as close to $\bar{u}$ as possible is an

alternative. Thus an **approximation error** or **modeling error** arises.

$$\varepsilon_1 = \|u_{\mathcal{F}}(x) - \bar{u}(x)\|_\Omega = \min_{u\in\mathcal{F}} \|u(x) - \bar{u}(x)\|_\Omega \tag{4.1}$$

This is an important indicator to measure the representation ability of the model.

Theorem 4.1 (Universal Approximation Theorem[32]). *The function family composed of MLP neural network is uniformly dense in the continuous function space.*

Namely for a given continuous objective function $\bar{u}(x)$, defined on a bounded closed set (compact set) $S \subset \mathbb{R}^n$, and for any given positive number $\varepsilon > 0$, there exists an MLP neural network with a single hidden layer $u(x)$, such that for all $x \in S$, there are $|\bar{u}(x) - u(x)| < \varepsilon$.

Theorem 4.2 (Pinkus best approximation error order of shallow neural network[33]). *Let B be the unit ball. Consider using a shallow MLP function $u : B \subset \mathbb{R}^n \to \mathbb{R}$ function to approximate a given function $\bar{u} : B \to \mathbb{R}$*

- *space of $\bar{u}$ is $B^{k,p} := \{\bar{u} \in W^{k,p}(B) \mid \|\bar{u}\|_{W^{k,p}} \leq 1\}$*
- *space of u is $\mathrm{R}_N := \{\sum_{i=1}^N u_i(\boldsymbol{a}_i^T \boldsymbol{x}) \mid \boldsymbol{a}_i \in \mathbb{R}^n,\ u_i \in C(\mathbb{R}),\ i = 1, \cdots, N\}$*

when $p \in [1, \infty], n \geq 2, k \geq 1$,

$$\sup_{\bar{u}\in B^{k,p}} \inf_{u\in \mathrm{R}_N} \|\bar{u} - u\|_{L^p} \leq C N^{\frac{k}{n-1}} \tag{4.2}$$

Theorem 4.3 (Maiorov best approximation order theorem[34]). *For $n \geq 2, k \geq 1$,*

$$\sup_{\bar{u}\in B^{k,2}} \inf_{u\in \mathrm{R}_N} \|\bar{u} - u\|_{L^2} \asymp -N^{\frac{k}{n-1}} \tag{4.3}$$

Combining [33] and [34] gives an estimate of the same order. Which means that,

$$C N^{\frac{k}{n-1}} \leq \varepsilon \Longleftrightarrow N \geq (\frac{C}{\varepsilon})^{\frac{n-1}{k}} \tag{4.4}$$

The width N of the shallow neural network needs to satisfy $N = \mathrm{O}((\frac{1}{\varepsilon})^{\frac{n-1}{k}})$. That is, when the width of the neural network is large enough, it can theoretically approach the real solution infinitely.

Since the KA representation theorem only explains the existence of $\psi^{p,q}$, we cannot ensure that it must be the form (3.27) to avoid the lack of smoothness of the inner function.

Theorem 4.4 (Lorentz [4]). *For continuous increasing function $\phi_q (q = 1, \cdots, 2n + 1)$, such that for any $f \in C([0, 1])^n$, there is a continuous function g, such that:*

$$f(x_1, \cdots, x_n) = \sum_{q=1}^{2n+1} g(\sum_{p=1}^{n} \lambda_p \phi_q(x_p)) \tag{4.5}$$

where $\lambda_p > 0 (p = 1, \cdots, n)$ *is* $\mathbb{Q}$ *independent, namely if rational number* $t_1, \cdots, t_n$ *satisfy* $\sum_{p=1} t_p \lambda_p = 0$*, then for* $\forall i, t_i = 0$

Model	**MLP**	**KAT**(ideal situation)
Theoretical basis	1943(McCulloch-Pitts)	1956(Komogrov et.al)
shallow expression	$\sum_{i=1}^{m} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$\sum_{q=1}^{2n+1} \varphi_q(\sum_{p=1}^{n} \psi^{p,q}(x_p))$
Theorem	(4.1)	(3.1)
Approximation Error	exist	not exist

Table 4.1: Theoretically speaking, KAT will be better than MLP in terms of approximation error.

This conclusion replaces $(2n+1)n$ inner-layer functions $\psi^{p,q}$ in the KA representation theorem with $2n+1$ inner-layer functions ϕ_q multiplied by n constants λ_p, which is an improved form of the KA representation theorem.

⌈Can be represented accurately⌋ and ⌈Has high-precision approximation⌋ are two completely different concepts. So KAN-model can't be simply explained by Komogrov-Arnold representation theorem.

Theorem 4.5 (KAN, Theorem 2.1[27])**.** *Assume* $f(x)(x \in \mathbb{R}^n)$ *has representation:*

$$f = (\Psi_{L-1} \circ \Psi_{L-2} \circ \cdots \Psi_1 \circ \Psi_0)(x) \tag{4.6}$$

where $\Psi_{l,i,j} \in C^{k+1}$*. Then there is only a constant* C *w.r.t* f*, so that there is the following error estimate related to the width* G *of the B-spline function: There is a* k*-order B-spline function* $\Psi^G_{l,i,j}$*, for any* $0 \le m \le k$*, there is*

$$\|f - ((\Psi^G_{L-1} \circ \Psi^G_{L-2} \circ \cdots \Psi^G_1 \circ \Psi^G_0)(x))\|_{C^m} \le C \cdot G^{-k-1+m} \tag{4.7}$$

Obviously, when G or k is sufficient big, the error will be sufficient small.

Model	**MLP**	**KAN**
	1943(McCulloch-Pitts)	2024(Liu et al.)
shallow expression	$\sum_{i=1}^{m} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$\sum_{q=1}^{2n+1} \varphi_q(\sum_{p=1}^{n} \underbrace{b_p(x_p) + spline_p(x_p)}_{\psi^{p,q}})$
Theorem	(4.1)	(4.5)
Approximation Error	exist	exist

Table 4.2: MLP has models first and then theory, while KAN has theory first and then models.

4.2 Generalization Error

When facing specific practical problems, data needs to be obtained by sampling from the definition domain of the problem.

- Let $T = \{(x_i, \bar{u}_i)\}_{i=1}^{n_{\text{data}}}$ be the set of all sample points.
- Loss $\mathcal{L}(\theta)$ depends on how we do sampling, so it can be write as $\mathcal{L}(\theta; T)$.
- Let the parameters θ that minimize the loss function be θ^*, and $u_T := u(\cdot; \theta^*)$.
- u_T and $u_{\mathcal{F}}$ may not be equal, so a generalization error arises.

$$\varepsilon_2 = \|u_T(x) - u_{\mathcal{F}}(x)\|_\Omega \tag{4.8}$$

4.3 Optimization Error

The parameters θ of the loss function $\mathcal{L}(\theta)$ require the use of optimization algorithms, such as Adam, SGD, L-BFGS, etc.

- $\mathcal{L}(\theta; T)$ is usually a non-convex function, and it is difficult to obtain a globally optimal parameter θ^*. Normally we can only obtain a local optimal parameter θ^*
- Let $u := u(\cdot; \theta^*)$
- u and u_T may not be equal, so a optimization error arises.

$$\varepsilon_3 = \|u(x) - u_T(x)\|_\Omega \tag{4.9}$$

Assume we have m sampling points $\{(x_i, \bar{u}_i)\}_{i=1}^m$, and loss function is defined in the form of $\mathcal{L}(\theta) = \frac{1}{\sqrt{m}} \sum_{i=1}^m \|u(x_i; \theta) - \bar{u}_i\|^2$.

According to gradient decent method $\theta_{k+1} = \theta_k - \eta \nabla L(\theta_k)$, we got:

$$\frac{\theta_{k+1} - \theta_k}{\eta} = -\nabla L(\theta_k) \tag{4.10}$$

when learning rate η is sufficient small, it can be regard as a continuous equation,

$$\frac{d}{dt}\theta = -\nabla L(\theta_k) \tag{4.11}$$

and this function is called "Gradient Flow". (4.10) can also be regarded as Euler forward difference discrete format of(4.11).

4.3.1 Optimization Error of MLP

Assume $\mathbf{u}(t) = [u(x_1; \theta(t)), \cdots, u(x_m; \theta(t))]$, $\mathbf{y} = [\bar{u}_1, \cdots, \bar{u}_m]$. Then $L(\theta) = \frac{1}{\sqrt{m}} \|\mathbf{u}(t) - \mathbf{y}\|_2^2$, it can be proved that,

$$\frac{d}{dt}\mathbf{u}(t) = -\mathbf{H}(t)((\mathbf{u}(t) - \mathbf{y})) \tag{4.12}$$

where $\mathbf{H}(t)$ is called **"Neural tangent kernel"** and defined as,

$$\mathbf{H} := \begin{bmatrix} H_{11} & H_{12} & \cdots & H_{1m} \\ H_{21} & H_{22} & \cdots & H_{2m} \\ \vdots & \vdots & & \vdots \\ H_{m1} & H_{m2} & \cdots & H_{mm} \end{bmatrix}, \quad H_{ij} = \langle \nabla u(x_i, \theta), \nabla u(x_j, \theta) \rangle \tag{4.13}$$

Theorem 4.6 (Exponential Convergence of Gradient Flow)**.** *If* $0 \leq s \leq t$, $\lambda_{min}(\boldsymbol{H}(s)) \geq \gamma > 0$,

$$\|\boldsymbol{u}(t) - \boldsymbol{y}\|^2 \leq e^{-\gamma t} \|\boldsymbol{u}(0) - \boldsymbol{y}\| \tag{4.14}$$

.

Proof. Since $\frac{d}{dt}\mathbf{u}(t) = -\mathbf{H}(t)((\mathbf{u}(t) - \mathbf{y}))$, so

$$\frac{d}{dt}\|\mathbf{u}(t) - \mathbf{y}\|^2 = \langle \mathbf{u}(t) - \mathbf{y}, \frac{d\mathbf{u}(t)}{dt} \rangle \tag{4.15}$$

$$= \langle \mathbf{u}(t) - \mathbf{y}, -\mathbf{H}(t)(\mathbf{u}(t) - \mathbf{y}) \rangle \tag{4.16}$$

$$\leq -\gamma \|\mathbf{u}(t) - \mathbf{y}\|^2 \tag{4.17}$$

Thus, $\|\mathbf{u}(t) - \mathbf{y}\|^2 \leq e^{-\gamma t}\|\mathbf{u}(0) - \mathbf{y}\|^2$. □[35]

[Du et al. 2018][35] has given: $u(x; \theta)$ is a Neural Network with one hidden layer, width is $n = \Omega(\frac{m^6}{\gamma_0^4 \delta^3})$ and activation function is ReLU, then with probability not less than $1 - \delta$, it holds that,

$$\lambda_{min}(\mathbf{H}(s)) \geq \gamma > 0 \tag{4.18}$$

where m is number of sampling points and $\gamma_0 > 0$ is minimal characteristic of matrix $\mathbf{H}^\infty$ and

$$\mathbf{H}^\infty := \mathbb{E}_\theta[\mathbf{H}(\theta)] \tag{4.19}$$

when $n \gg m$, we say the neural network is **"Over - Parameterized"**.

Existing optimization error analyses for multilayer perceptron (MLP) neural networks rely on over-parameterization assumptions(4.18), based on high probability condition.

Spectral Bias: proposed that when the neural network width is sufficiently large, the NTK[36](4.13) matrix does not change much.

Let $\mathbf{H}(t) = H^*$ namely H^∞, note that $\mathbf{y}$ independent of t, then

$$\frac{d}{dt}(\mathbf{u}(t) - \mathbf{y}) \approx -\mathbf{H}^* \cdot (\mathbf{u}(t) - \mathbf{y}) \tag{4.20}$$

Solve the ODE we got,

$$\mathbf{u}(t) - \mathbf{y} \approx e^{-\mathbf{H}^* t}(\mathbf{u}(0) - \mathbf{y}) \tag{4.21}$$

Since $\mathbf{H}^*$ is symmetric positive definite, it can be decomposed as $\mathbf{H}^* = \mathbf{Q}\Lambda\mathbf{Q}^T$,

$$\mathbf{u}(t) - \mathbf{y} \approx \mathbf{Q}e^{-\Lambda t}\mathbf{Q}^T(\mathbf{u}(0) - \mathbf{y}) \tag{4.22}$$

Let the eigenvector of $\mathbf{H}^*$ corresponding to the eigenvalue λ_i be $\mathbf{q}_i$, and $\mathbf{Q}$ is orthogonal then

$$\begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_m^T \end{bmatrix} (\mathbf{u}(t) - \mathbf{y}) = \begin{bmatrix} e^{-\lambda_1 t} & & & \\ & e^{-\lambda_2 t} & & \\ & & \ddots & \\ & & & e^{-\lambda_m t} \end{bmatrix} \begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_m^T \end{bmatrix} (\mathbf{u}(0) - \mathbf{y}) \tag{4.23}$$

Note that the larger λ_i is, the faster it will converge in the $\mathbf{q}_i$ direction; we call the ones with larger λ the low-frequency components and the smaller λ the high-frequency components, namely low-frequency components converge faster than high-frequency components. This is the phenomenon of "**spectral bias**".

One possible approach is to reduce the spectral condition number $\frac{\lambda_{\max}}{\lambda_{\min}}$ of $\mathbf{H}^*$, i.e., to perform Fourier feature embedding[37]. Its main idea is to use a kernel function to map the input coordinates to high-frequency signals before passing through the MLP.

$$\gamma(\mathbf{x}) = \begin{bmatrix} \cos(\mathbf{B}\mathbf{x}) \\ \sin(\mathbf{B}\mathbf{x}) \end{bmatrix}, \quad \gamma : \mathbb{R} \to \mathbb{R}^m. \tag{4.24}$$

where each entry in $\mathbf{B} \in \mathbb{R}^{m \times d}$ is sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ and $\sigma > 0$ is a user-specified hyperparameter.

4.3.2 Optimizaition Error of KAN

Spectral Bias in KAN[38]: Define a KAN as $\mathrm{KAN}(\cdot, \theta) : \mathbb{R}^d \to \mathbb{R}^{d'}$,

$$\mathbf{KAN}(\mathbf{x}; \theta)_i = \sum_{j=1}^{d} \sum_{l=1}^{G+k-1} c_{ijl} B_l(x_j), \quad i = 1, \cdots, d' \tag{4.25}$$

where B_l is B-spline with grid size G, order k.

Consider the loss $\mathcal{L}$ below,

$$L(\theta) = \int_\Omega \|f(\mathbf{x}) - \mathrm{KAN}(\mathbf{x}; \theta)\|^2 d\mathbf{x} \tag{4.26}$$

with discrete form

$$\frac{1}{m} \sum_{i=1}^{m} \|f(\mathbf{x}_i) - \mathrm{KAN}(\mathbf{x}_i; \theta)\|^2 \tag{4.27}$$

where $\theta = \{c_{ijl}\}$, so $L(\theta)$ is the quadratic form of θ,

$$L(\theta) = \frac{1}{2}\theta^T M \theta + b^T \theta \tag{4.28}$$

and $M = \{M_{(i,j,l),(i',j',l')}\}(i = 1, \cdots, d'; j = 1, \cdots, d; l = 1, \cdots, G + k - 1)$.

$$M_{(i,j,l),(i',j',l')} = \begin{cases} \int_\Omega B_l(x_j) B_{l'}(x_{j'}) \, \mathrm{d}\mathbf{x}, & \text{if } i = i', \\ 0, & \text{if } i \neq i'. \end{cases} \tag{4.29}$$

Note: $N = (G + k - 1)dd'$, M is an N-dimensional symmetric matrix where d is input size and d' is output size.

Theorem 4.7 (Theorem 4.1[38]). *Let the eigenvalues of M be $o \leq \lambda_1 \leq \cdots \lambda_N$. These exists a constant C s.t.*

$$\frac{\lambda_N}{\lambda_{d'(d-1)+1}} \leq Cd \tag{4.30}$$

Constant C only depends on the order of the spline function.

Notably, with the exception of $(d-1)d'$ eigenvectors, the spectral condition number of M maintains a consistent upper bound, unaffected by the hyperparameters G and k of the spline function. This non-uniqueness in representation underscores a key theoretical advantage of KAN over MLP in spectral bias analysis.

Chapter 5

Problem Setup

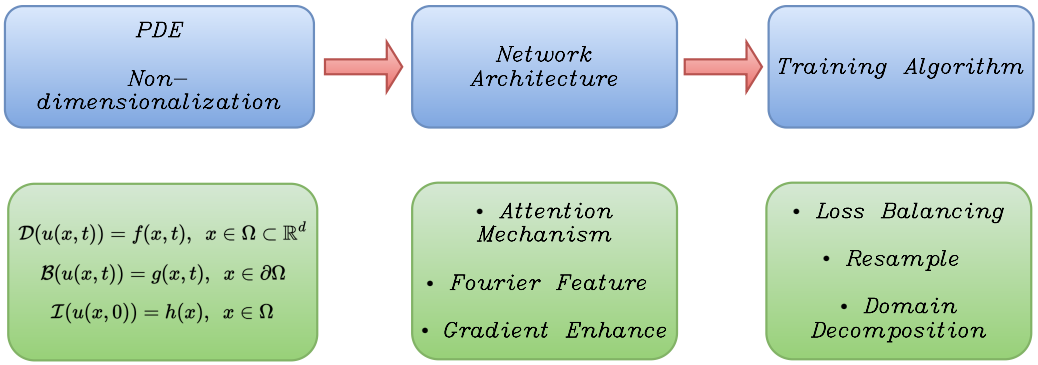


Figure 5.1: When embarking on a PINN problem, the need for nondimensionalization is first decided based on the scale of the physical quantity. When the input and output of the neural network are within a reasonable range, the convergence becomes faster and the results are stable. An appropriate network architecture can then be selected based on the complexity of the problem. It has been proven in the previous chapter that Fourier feature embedding can effectively reduce spectral deviations. Later in the training phase we can introduce some loss balancing, adaptive resampling or local augmentation.

Algorithm 1 Training Pipeline

1: Define the PDE system and design the sampling strategy.
   (a) Sampling, whether uniformly distributed, Latin hypercube sampling or grid sampling, is to obtain sample points uniformly in the spatial and temporal domain.
   (b) Local-Enhance: According to [5, 6], the change is mainly concentrated at interphase-boundary, and this part can be additionally sampled for local enhancement.
2: Non-dimensionalize the PDE system.
3: Choose a neural network architecture.
4: Pass in the input data directly, or perform random Fourier feature (RFF) embedding on the input data (optional). Here we take the initiative to add a non-negative layer before the output based on the reality of the diffusion reaction equation, strictly limiting the concentration not to 0.
5: Based on the output of the neural network, construct the physical information part of the problem. Formulate the weighted loss function based on the PDE equations (3.12).
6: Use the N-step gradient descent algorithm to update the neural network parameters θ, or set a loss threshold ε,
7: **for** $n = 1, \cdots, N$ **do**
   (a) Update the parameters θ via gradient descent:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n), \tag{5.1}$$

   (b) Residual adaptive resampling (Optional): Re-randomizing sampling in the computational domain $\{x_{\mathrm{ic}}^i\}_{i=1}^{N'_{ic}}$, $\{x_{\mathrm{bc}}^i, t_{\mathrm{bc}}^i\}_{i=1}^{N'_{bc}}$ and $\{x_{\mathrm{r}}^i, t_{\mathrm{r}}^i\}_{i=1}^{N'_{r}}$. Evaluate each loss term and select k-points with k-biggest loss values. Add them into current data collocation points for next iteration.
8: **end for**

5.1 Nondimensionalization

Data normalization[39, 40] is a well-established pre-processing technique in traditional deep learning, typically involving feature scaling to ensure comparable magnitudes and ranges. However, for Physics-Informed Neural Networks (PINNs), particularly in forward PDE problem solving, this standard approach presents challenges due to the unavailability of target solutions.

In such scenarios, ensuring that output variables remain within a reasonable range becomes critical. **Non-dimensionalization** emerges as an elegant solutiona powerful technique widely employed in mathematics and physics to simplify and analyze complex systems. This approach involves transforming the original system into an equivalent dimensionless representation by selecting fundamental units or characteristic values, effectively using re-scaled variables to simplify convergence.

The significance of this method stems from several key considerations:

- **Improved numerical stability:** Different variables in physical systems (such as time, space coordinates, temperature, velocity, etc.) often have different magnitudes. Dimensionless transformation can convert these variables to similar scales, thus avoiding the situation where certain variables dominate the loss function and helping to improve the stability of numerical calculations.
- **Gradient consistency:** When all input and output variables are of a similar order of magnitude, the network parameters are updated more evenly, preventing optimization difficulties caused by too large or too small gradients in certain directions. This consistency is particularly important for gradient descent algorithms as it ensures that each weight is adjusted in the same proportion, thus speeding up convergence and reducing the risk of local minima.
- **Avoid extreme changes:** This method helps create a smoother loss function surface, reduces the impact of local minima, and promotes the search for global optimal solutions. Bringing the prediction of initial conditions closer to the true solution improves the accuracy of the dynamically evolving system.

5.2 Nondimensionalization Application

Parameters	Numerical value
SiC-Substrate Thickness L	600nm
Meta-Film Thickness h	100nm
Annealing Time t	0.1/10/60 min
Diffusivity of Nickel D_{Ni} of (2.19)	$360\mathrm{nm}^2/\mathrm{min}$
Reaction Coefficient k_{11} of (2.10)	$3.6 \times 10^{-3}\mathrm{nm}^3/\mathrm{min}$
Reaction Coefficient k_{12} of (2.12)	$3.6 \times 10^{-3}\mathrm{nm}^3/\mathrm{min}$

Table 5.1: System of differential equations symbols with given values where initially given value are $D_{\mathrm{Ni}} = 6 \times 10^{-14}\mathrm{cm}^2/\mathrm{s}$ and $k_{11} = k_{12} = 6 \times 10^{-26}\mathrm{cm}^3/\mathrm{s}$ To simplify the calculations, we first convert the units in the table above to nanometers and minutes.

Then the equivalent system in the form of nondimensionalized equations in lieu of (2.13) (2.14), (2.15),(2.16) (2.17),(2.18) ,(2.20) (2.21),(2.22), and (2.23) is given by:

Take Nickel for instance:

$$\frac{\partial C_{\mathrm{Ni}}}{\partial t} = \frac{\partial}{\partial x}(D^* \frac{\partial C_{\mathrm{Ni}}}{\partial x}) - k_{11} C_{\mathrm{Ni}} C_{\mathrm{SiC}} - k_{12} C_{\mathrm{Ni}} C_{\mathrm{NiSi}} \tag{5.2}$$

Characteristic Concentration	Numerical value
Silicon-atom Concentration: C^*_{SiC}	$4.8 \times 10^{22}\mathrm{cm}^{-3} = 48\mathrm{nm}^{-3}$
Metal-atom concentration: C^*_{Ni}	$9 \times 10^{22}\mathrm{cm}^{-3} = 90\mathrm{nm}^{-3}$
C^*_{NiSi}	$90\mathrm{nm}^{-3}$
$C^*_{\mathrm{NiSi_2}}$	$90\mathrm{nm}^{-3}$
$C^*_{\mathrm{Ni_2Si}}$	$90\mathrm{nm}^{-3}$
C^*_{C}	$90\mathrm{nm}^{-3}$

Table 5.2: Selection of Characteristic Concentration. Where C^*_{SiC}: intrinsic siliconatom concentration in the substrate and N_{Ni}:intrinsic metal atom concentration, are pre-given. The other characteristic concentrations are reasonably chosen to be the same as C^*_{Ni}.

The original concentration will be expressed as the product of the scaled concentration and the characteristic concentration with

$$C_\alpha = C'_\alpha \cdot C^*_\alpha, \quad \alpha =\in \{\mathrm{Ni}, \mathrm{SiC}, \cdots, \mathrm{Ni_2Si}\} \tag{5.3}$$

$$t = t' \cdot T, \quad x = x' \cdot L. \tag{5.4}$$

Non-dimensionalized Function:

$$\begin{cases} \frac{\partial C_{\mathrm{Ni}}}{\partial t} = \frac{\partial C_{\mathrm{Ni}} C^*_{\mathrm{Ni}}}{\partial t' T}, \\ \frac{\partial}{\partial x}(D^* \frac{\partial C_{\mathrm{Ni}}}{\partial x}) = \frac{\partial}{\partial x' L}(D_{\mathrm{Ni}} \underbrace{\frac{C'_{\mathrm{SiC}} C^*_{\mathrm{SiC}} + \cdots + C'_{\mathrm{Ni_2Si}} C^*_{\mathrm{Ni_2Si}}}{C'_{\mathrm{Ni}} C^*_{\mathrm{Ni}} + \cdots + C'_{\mathrm{Ni_2Si}} C^*_{\mathrm{Ni_2Si}}}}_{(*)} \cdot \frac{\partial C_{\mathrm{Ni}} C^*_{\mathrm{Ni}}}{\partial x' L}), \\ -k_{11} C_{\mathrm{Ni}} C_{\mathrm{SiC}} = -k_{11} C'_{\mathrm{Ni}} C^*_{\mathrm{Ni}} C'_{\mathrm{SiC}} C^*_{\mathrm{SiC}}, \\ -k_{21} C_{\mathrm{Ni}} C_{\mathrm{NiSi}} = -k_{21} C'_{\mathrm{Ni}} C^*_{\mathrm{Ni}} C'_{\mathrm{NiSi}} C^*_{\mathrm{NiSi}}. \end{cases} \tag{5.5}$$

Get the scaled representation:

$$\frac{\partial C^*_{\mathrm{Ni}}}{\partial t'} = \frac{\partial}{\partial x'}[D_{\mathrm{Ni}} \frac{T}{L^2}(*) \cdot \frac{\partial C_{\mathrm{Ni}} C^*_{\mathrm{Ni}}}{\partial x' L}] - \frac{k_{11} T}{C^*_{\mathrm{Ni}}} C'_{\mathrm{Ni}} C^*_{\mathrm{Ni}} C'_{\mathrm{SiC}} C^*_{\mathrm{SiC}} - \frac{k_{21} T}{C^*_{\mathrm{Ni}}} C'_{\mathrm{Ni}} C^*_{\mathrm{Ni}} C'_{\mathrm{NiSi}} C^*_{\mathrm{NiSi}} \tag{5.6}$$

Here the diffusion coefficient and reaction coefficient are re-expressed like $D* \to D * \frac{T}{L^2}$ and $k \to k \frac{C^*_A C^*_B}{C^*_A}$ and be rescaled to a similar scale.

Naturally there will be some corresponding small changes in the initial and boundary conditions.

$$\frac{\partial C'_{\mathrm{Ni}} C^*_{\mathrm{Ni}}}{\partial x' L}(x,t) = 0, \quad x = 0 \wedge x = L \tag{5.7}$$

$$\frac{\partial C'_{\mathrm{Ni}}}{\partial x'}(x,t) = 0 \cdot \frac{L}{C^*_{\mathrm{Ni}}} = 0, \quad x = 0 \wedge x = L \tag{5.8}$$

$$C'_{\mathrm{Ni}}C^*_{\mathrm{Ni}}(x,0) = \begin{cases} C^*_{\mathrm{Ni}}, & 0 \le x \le h \\ 0, & h < x \le L \end{cases} \Longrightarrow C'_{\mathrm{Ni}}(x,0) = \begin{cases} \frac{C^*_{\mathrm{Ni}}}{C^*_{\mathrm{Ni}}}, & 0 \le x \le \frac{h}{L} \\ 0, & \frac{h}{L} < x \le 1 \end{cases} \tag{5.9}$$

$$C'_{\mathrm{SiC}}C^*_{\mathrm{SiC}}(x,0) = \begin{cases} C^*_{\mathrm{SiC}}, & 0 \le x \le h \\ 0, & h < x \le L \end{cases} \Longrightarrow C'_{\mathrm{SiC}}(x,0) = \begin{cases} \frac{C^*_{\mathrm{SiC}}}{C^*_{\mathrm{SiC}}}, & 0 \le x \le \frac{h}{L} \\ 0, & \frac{h}{L} < x \le 1 \end{cases} \tag{5.10}$$

After the above nondimensional transformation, the effective heterogeneous diffusivity for the annealing time 60min is scaled to,

$$D^* = \frac{360\mathrm{nm}^2/\mathrm{min} \cdot 60\mathrm{min}}{600\mathrm{nm}^2} \longrightarrow 0.06 \tag{5.11}$$

The reaction coefficient dominated by SiC is scaled to:

$$k' = \frac{3.6 \times 10^{-3}\mathrm{nm}^3/\mathrm{min} \times 60\mathrm{min}}{48\mathrm{nm}^{-3}} = 0.0045 \tag{5.12}$$

And reaction coefficient dominated by Ni is scaled to:

$$k^* = \frac{3.6 \times 10^{-3}\mathrm{nm}^3/\mathrm{min} \times 60\mathrm{min}}{90\mathrm{nm}^{-3}} = 0.0024 \tag{5.13}$$

Chapter 6

Results

6.1 Function Fitting

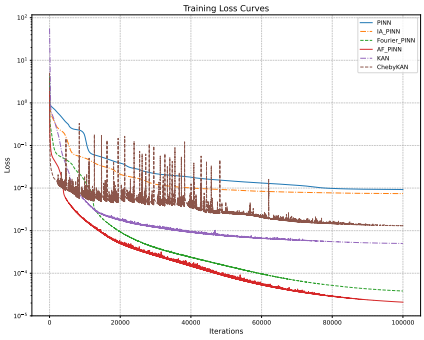
When exploring the vast field of neural networks, we often start with basic concepts and simple applications. A good starting point is to understand the performance of different network structures through function fitting, This not only provides us with an intuitive perspective to observe the learning ability of the model, but also provides a solid foundation for complex tasks such as physically informed neural networks (PINNs).

First consider a simple function fitting problem: Given a set of input-output pairs (x_i, y_i), the goal is to train a neural network to learn the mapping $y = f(x)$ hidden behind this.

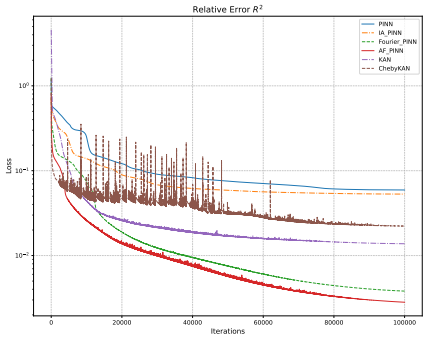
$$\begin{cases} f_1(x) = \sin(2.5\pi x) + x^2, & x \in [0, 1.5) \\ f_2(x) = 0.5xe^{-x} + |\sin(5\pi x)|, & x \in [1.5, 3) \\ f_3(x) = \log_2(x-1) - \cos(2\pi x), & x \in [3, 4.5) \\ f_4(x) = \lfloor \frac{(x-4.5)}{0.3} \rfloor, & x \in [4.5, 6) \end{cases} \tag{6.1}$$

Mode	Loss	Relative Error	Training Time
MLP	9.265e-03	0.0593	04:05
Attention-MLP	7.411e-03	0.0530	06:40
Fourier-MLP	3.832e-05	0.0038	04:47
AF-MLP	2.095e-05	0.0028	07:38
KAN	5.006e-04	0.0138	28:23
ChebyKAN	1.305e-03	0.0223	11:19

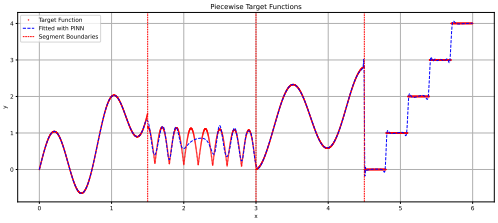
Table 6.1: For this problem we use MLP: [32]×3 and KAN: [32]×3. Use the same training epochs: 100000, initial learning rate 1×10^{-4}, end learning rate 1×10^{-6}, and adopt cosine annealing learning rate scheduler.



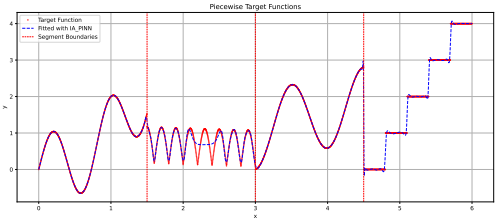
(a) Loss Curve for Segmentation Function



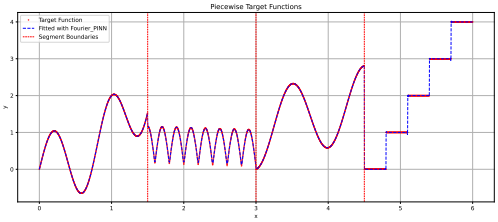
(b) Loss Curve for Segmentation Function



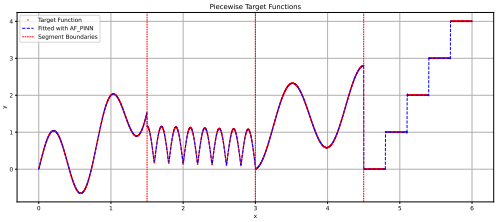
(a) MLP



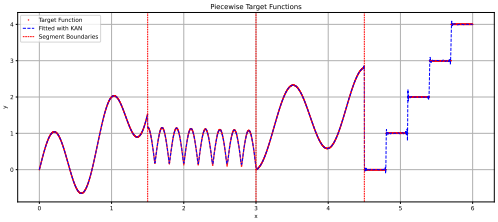
(b) Attention MLP



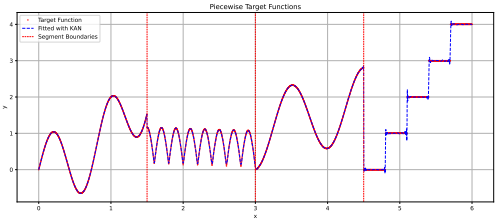
(c) Fourier MLP



(d) Attention Fourier MLP



(e) KAN



(f) Chebyshev Polynomial KAN

Figure 6.2: Segmentation function (6.1)

Obviously for MLP and Attention MLP, the fitting effect is poor in the part where the function changes rapidly. And for KAN and ChebyKAN, the results are improved in the fast-changing area. After embedding random Fourier features into MLP, not only can it fully learn the fast-changing area, but also get better fitting results in the step function part where other networks are not good at.

6.2 Simple Problem: Burgers Equations

After solving a basic function fitting problem and understanding the performance of different network architectures, we now tackle the a little bit more complex Burgers equation using our previously discussed networks. In the original PINN paper[9] by Raissi et al., Burgers' equation was used as an intuitive example to discuss how to use PINNs to solve partial differential equations. It is a nonlinear partial differential equation that is used in fluid mechanics to model the one-dimensional flow of viscous fluids and is also a simple model for studying nonlinear and diffusion phenomena.

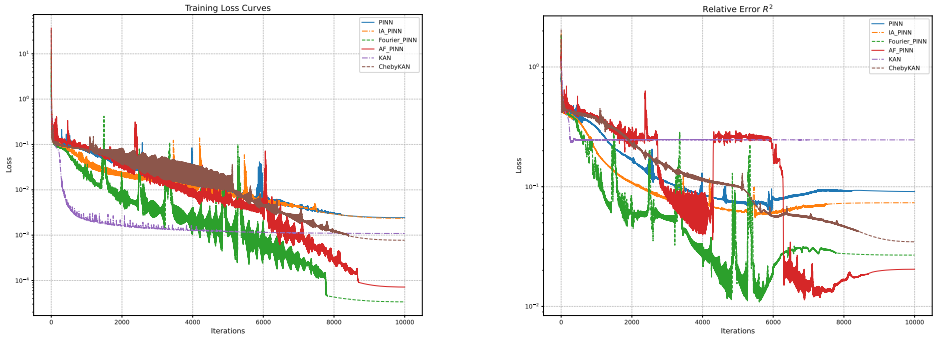
$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0.01\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \quad t \in [0, 1]. \tag{6.2}$$

with the Dirichlet boundary conditions and initial conditions:

$$\begin{cases} u(-1, t) = u(1, t) = 0 \\ u(x, 0) = -\sin(\pi x) \end{cases} \tag{6.3}$$

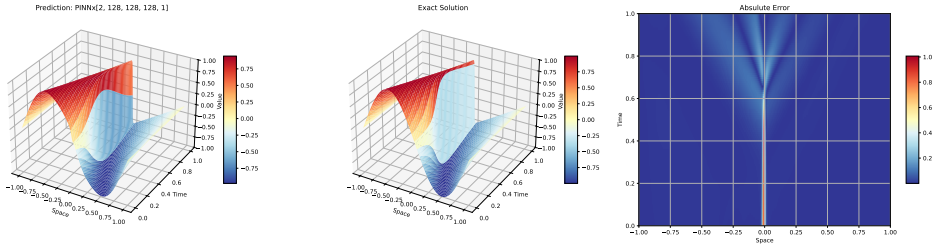
Mode	Loss	Relative Error	Training Time
MLP	2.430e-03	0.09111	03:45
Attention-MLP	2.348e-03	0.07347	08:51
Fourier-MLP	3.383e-05	0.02683	05:10
AF-MLP	7.168e-05	0.02044	11:58
KAN	1.089e-03	0.24607	28:44
ChebyKAN	7.719e-04	0.03465	15:10

Table 6.2: Burgers Equations: The training epochs is 10000, where the size of MLPs are $[128] \times 3$, the size of KAN is $[16] \times 3$, and the size of ChebyKAN is $[64] \times 3$.

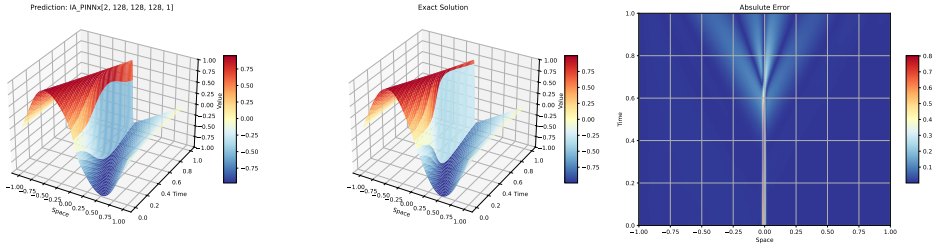


(a) Loss curve of Burgers Equations (b) Error curve of Burgers Equations

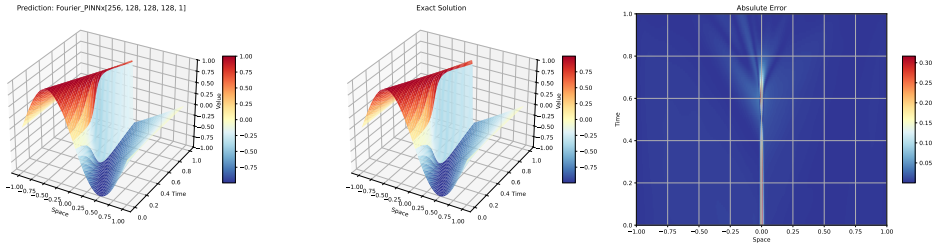
Figure 6.3: It is observed that the loss of KAN decreases rapidly at the beginning of training and converges quickly. But the relative error does not change. Considering the trade-off between training time and training results, applying Fourier transform on the input is a simple and effective method.



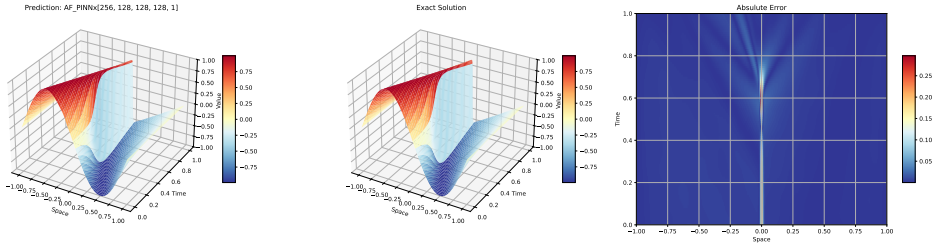
(a) MLP



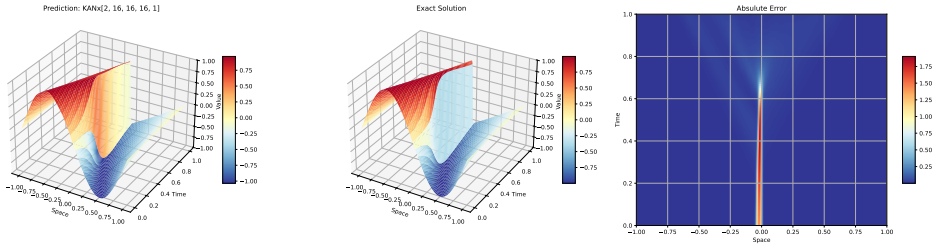
(b) Attention MLP



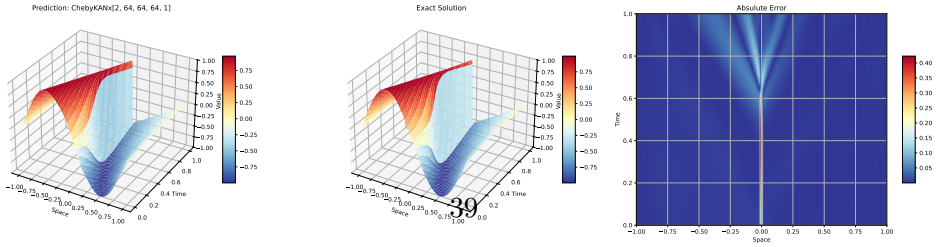
(c) Fourier MLP



(d) Attention Fourier MLP



(e) KAN



(f) Chebyshev Polynomial KAN

Figure 6.4: Burgers equation (6.2) (6.3)

For the Burgers equation, its solution shows rapid changing characteristics near the position x=0, and this area has therefore become the main source of errors. As the dynamic evolution process progresses, that is, as time increases, the small initial errors will gradually accumulate and be amplified, which further affects the overall accuracy of the understanding.

6.3 Diffusion Reaction System of Nickel and Silicon Carbide

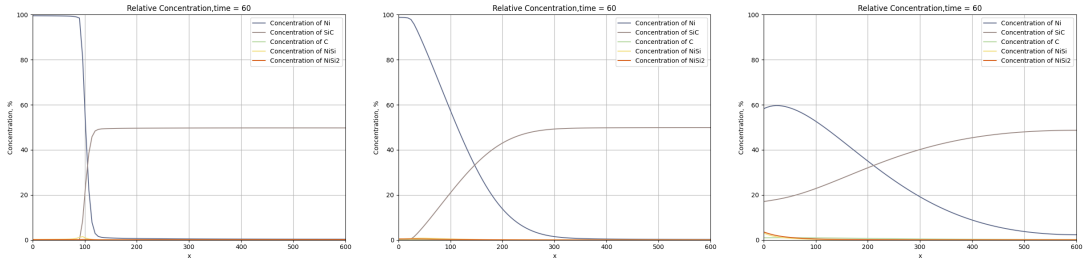
After completing the transition from function fitting to simple PINN application, we finally focus on the diffusion reaction equations of Ni-SiC. It's like walking along a well-paved road. Each step builds on previous knowledge, gradually deepening the understanding of how neural networks work, and ultimately achieving a silky transition from basic concepts to advanced applications.

0.1min	10min	60min
1×10^{-4}	1×10^{-2}	6×10^{-2}

Table 6.3: Upper bound of diffusivity for different annealing time

6.3.1 Diffusion-Part

Metal diffuses from the deposited film to the silicon carbide substrate. According to the Kirkendall effect, the interphase or the plane where the relative concentrations of metal and silicon are equal (i.e., the Kirkendall plane) moves toward the surface in this case. Metal concentrations exhibit a roughly symmetrical distribution (shown below).



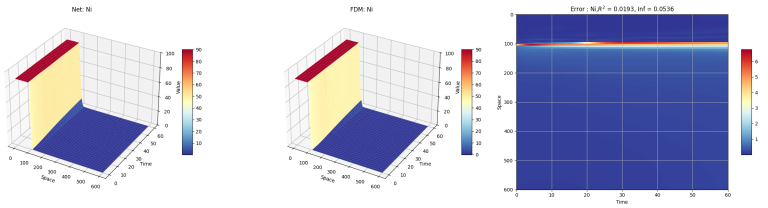
(a) t = 0.1 (b) t = 10 (c) t = 60

Figure 6.5: Here I take Attention Fourier PINN as an example. It can be seen that after the intersection of Ni and SiC, the relative concentrations between them present a symmetrical structure. Also as the annealing time increases, due to problems such as neural network initialization, errors accumulated. Concentrations of other components that should not exist appear at $x = 0$.

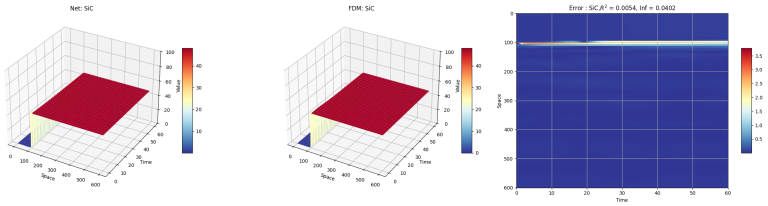
In the results below, the MLP is trained for $10,000$ epochs; the KAN is trained for $2,500$ epochs; and the ChebyKAN is also trained for $10,000$ epochs.

	Ni	SiC	Loss	Running Time
PINN	0.033	0.008	2.851e-03	673.70s
IA-PINN	0.02	0.006	1.787e-03	1455.45s
F-PINN	0.023	0.008	8.020e-04	984.61s
AF-PINN	0.019	0.005	1.035e-03	1867.30s
KAN	0.042	0.014	4.677e-03	1553.49s
ChebyKAN	0.034	0.013	2.739e-03	2402.41s

Table 6.4: Annealing Time = 0.1 min, set MLP to be $[64] \times 3$ and set KAN to be $[16] \times 3$.



(a) Ni: t = 0.1 min

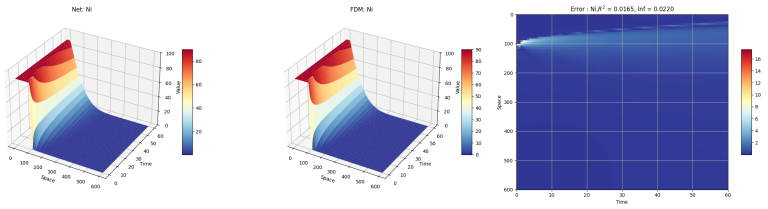


(b) SiC: t = 0.1 min

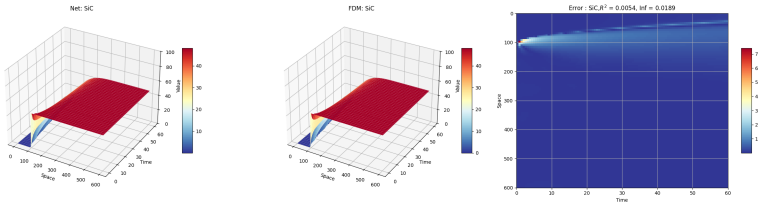
Figure 6.6: Error mainly appeared in the interphase-boundary. AF-PINN has a better performance than others.

	Ni	SiC	Loss	Running Time
PINN	0.052	0.018	4.799e-03	704.34s
IA-PINN	0.027	0.017	2.437e-03	1689.29s
F-PINN	0.027	0.011	1.861e-03	1208.88s
AF-PINN	0.016	0.005	1.741e-03	2586.36s
KAN	0.014	0.008	4.408e-03	2658.56s
ChebyKAN	0.033	0.018	3.157e-03	2436.28s

Table 6.5: Annealing Time = 10 min, set MLP to be $[96] \times 3$ and set KAN to be $[24] \times 3$.



(a) Ni: t = 10 min



(b) SiC: t = 10 min

Figure 6.7: The error is mainly concentrated at the initial condition interphase boundary and accumulates over time.

	Ni	SiC	Loss	Running Time
PINN	0.053	0.009	8.085e-03	807.22s
IA-PINN	0.023	0.009	5.294e-03	2108.14s
F-PINN	0.036	0.008	4.511e-03	1627.85s
AF-PINN	0.031	0.008	3.683e-03	3473.01s
KAN	0.035	0.016	8.140e-03	3449.32s
ChebyKAN	0.04	0.016	6.539e-03	2602.55s

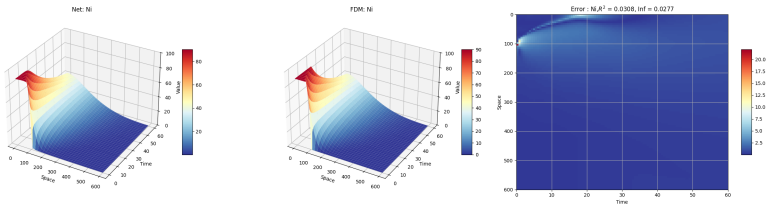
Table 6.6: Annealing Time = 60 min, set MLP to be $[128] \times 3$ and set KAN to be $[32] \times 3$.

In the process of diffusion only, PINN with Attention + random Fourier features always performs best. Although it takes the longest time to train among all the neural network architectures using MLP.

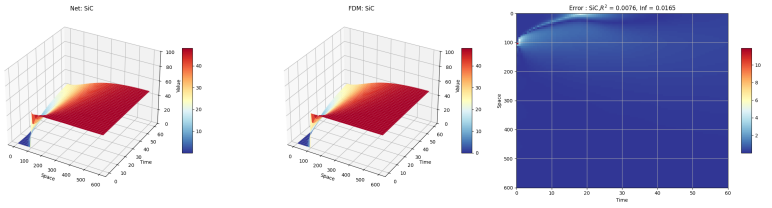
But here we noticed an interesting phenomenon, that is, in all the absolute error graphs, the error is always significant at the interphase change part, that is $x = 100$, and it mainly focuses on the initial part. Therefore, I infer that for a dynamically evolving system, the difficulty in learning the initial conditions will lead to the accumulation of errors in subsequent times.

6.3.2 Diffusion Reaction System

For each size the first row is L^2 error, second row is L^∞ Error.



(a) Ni: t = 60 min

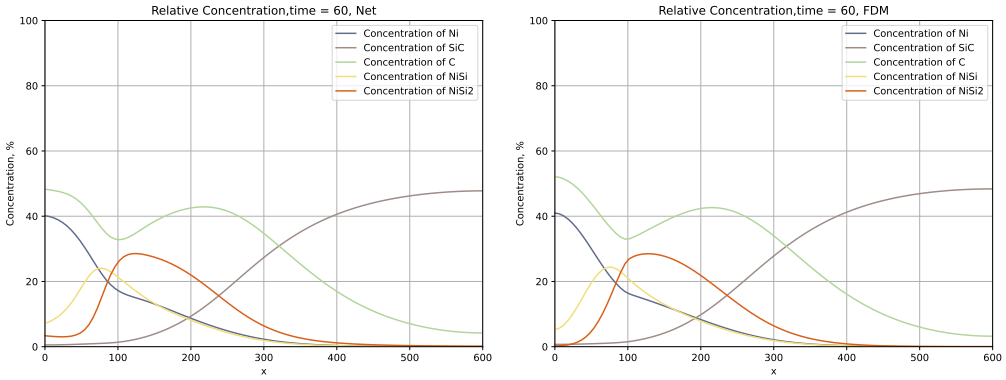


(b) SiC: t = 60 min

Figure 6.8: Similar to when $t = 10$, the error is mainly concentrated at the initial condition interphase boundary and accumulates over time.

Size	Ni	SiC	C	NiSi	NiSi2	Mean Error	Loss
[64] × 3	0.047	0.007	0.03	0.082	0.076	0.04830	3.491e-03
3455.90s	0.048	0.012	0.072	0.126	0.109	0.07357	/
[96] × 3	0.044	0.007	0.032	0.085	0.073	0.04798	2.719e-03
3368.19s	0.05	0.011	0.077	0.125	0.152	0.08117	/
[128] × 3	0.047	0.006	0.036	0.065	0.068	0.04453	2.246e-03
3399.29s	0.077	0.019	0.089	0.082	0.118	0.06743	/
[64] × 4	0.067	0.007	0.033	0.101	0.068	0.05508	1.377e-03
3876.97s	0.105	0.007	0.073	0.172	0.105	0.09245	/
[96] × 4	0.04	0.007	0.03	0.103	0.082	0.05235	1.009e-03
3796.86s	0.039	0.007	0.069	0.178	0.142	0.08692	/
[128] × 4	0.051	0.01	0.039	0.061	0.092	0.05069	3.635e-04
3992.33s	0.081	0.014	0.104	0.096	0.145	0.08802	/
[64] × 5	0.282	0.029	0.086	0.236	0.132	0.15325	8.028e-04
4384.06s	0.355	0.039	0.117	0.291	0.125	0.18531	/
[96] × 5	0.186	0.023	0.068	0.179	0.122	0.11537	1.676e-04
4356.85s	0.25	0.031	0.127	0.23	0.153	0.15829	/
[128] × 5	0.383	0.05	0.152	0.316	0.234	0.22684	5.764e-05
4570.19s	0.451	0.064	0.16	0.278	0.293	0.24899	/
[64] × 6	0.037	0.01	0.042	0.114	0.106	0.06173	7.577e-04
4869.44s	0.053	0.03	0.108	0.181	0.179	0.10693	/
[96] × 6	0.103	0.022	0.062	0.105	0.137	0.08564	3.426e-05
4877.49s	0.152	0.01	0.076	0.111	0.193	0.11249	/
[128] × 6	0.303	0.029	0.077	0.279	0.146	0.16658	7.297e-04
5250.34s	0.375	0.037	0.128	0.37	0.199	0.22200	/

Table 6.7: PINN: In the case where the activation function is Tanh, training epochs are set to 50,000. Under [128] × 3 got the best result among all configurations. Although the results show that the loss decreases as the size increases, the relative error does not actually minimize, but instead shows signs of overfitting.



(a) Relative concentration of PINN in size [128] × 3

(b) Relative concentration of FDM as reference

Figure 6.9: Relative concentration comparison of PINN and FDM at t=60.

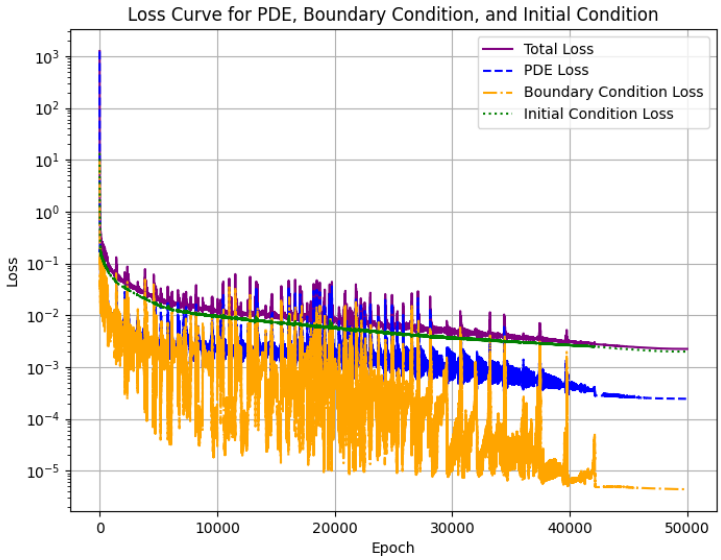
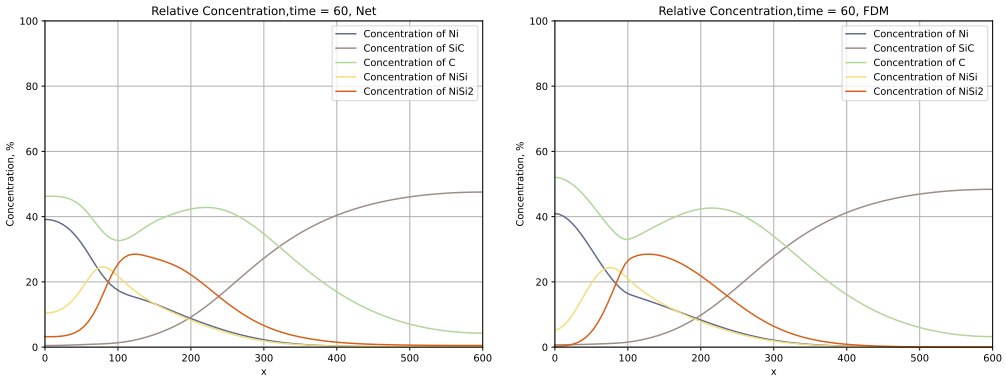


Figure 6.10: Loss curve of PINN in size [128] × 3

During the training process, it can be noticed that the loss in the initial condition part always dominates the total loss. In the early stages of training, the loss decreases rapidly. The initial condition loss then becomes almost linear with a fixed slope. The PDE loss and boundary condition loss oscillate down during the training process. After using the cosine annealing learning rate scheduler, the loss curve area stabilizes as the learning rate gradually decreases at the end of training.

Size	Ni	SiC	C	NiSi	NiSi2	Mean Error	Loss
[32] × 3	0.067	0.012	0.042	0.125	0.095	0.0682	7.77E-03
8593.05s	0.064	0.021	0.069	0.171	0.145	0.094	/
[64] × 3	0.06	0.009	0.049	0.129	0.119	0.0732	6.13E-03
16590.32s	0.057	0.019	0.101	0.2	0.245	0.1244	/
[32] × 4	0.052	0.006	0.034	0.119	0.081	0.0584	3.91E-03
11345.04s	0.052	0.012	0.075	0.2	0.135	0.0948	/
[64] × 4	0.046	0.005	0.027	0.114	0.114	0.0612	4.04E-03
21986.05s	0.045	0.045	0.072	0.199	0.26	0.1242	/
[32] × 5	0.046	0.005	0.042	0.128	0.105	0.0652	3.22E-03
14077.26s	0.043	0.01	0.107	0.209	0.192	0.1122	/
[64] × 5	0.042	0.005	0.034	0.109	0.107	0.0668	2.93E-03
27413.64s	0.042	0.01	0.08	0.218	0.279	0.1258	/

Table 6.8: KAN: Based on the implementation of Efficient-KAN, the inner function is bspline + silu. Training epochs are set to 10,000. Under [32] × 4 got the best result among all configurations. For KAN, the relative error of the NiSi part is difficult to reduce. This is the main reason for the poor results. In addition, the long training time of KAN is an important reason why it is difficult to put into practical application. Also notice that as the size increases, the loss decreases, and the relative error does not decrease significantly.



(a) Relative concentration of KAN in size [32] × 4

(b) Relative concentration of FDM as reference

Figure 6.11: Relative concentration comparison of KAN and FDM at t=60.

Also in the early stages of training, the loss drops rapidly, the initial condition loss dominates the total loss, and the PDE loss and boundary condition loss oscillate violently. In the area $x \in [0, 50]$, the relative concention for Carbon is lower than benchmark, and the relative concentrations of NiSi and $NiSi_2$ are higher than benchmark.

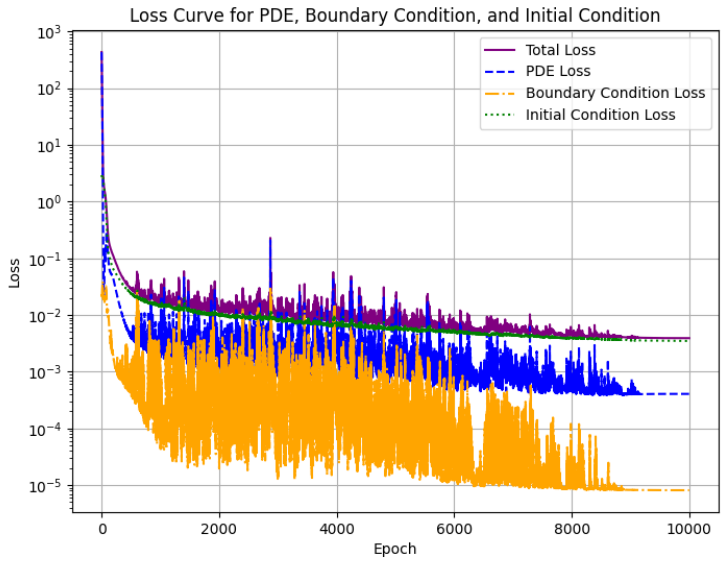
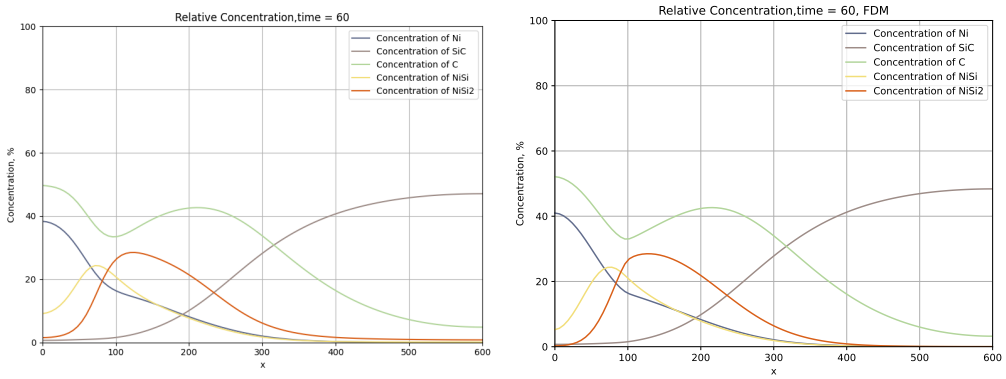


Figure 6.12: Loss curve of KAN in size [32] × 4

Size	Ni	SiC	C	NiSi	NiSi2	Mean Error	Loss
[64] x 3	0.044	0.015	0.066	0.134	0.124	0.0766	1.14E-03
9745.85s	0.051	0.018	0.127	0.195	0.146	0.1074	/
[128] x 3	0.036	0.008	0.04	0.091	0.097	0.0544	5.14E-04
17556.42s	0.039	0.008	0.1	0.142	0.127	0.0832	/
[64] x 4	0.038	0.012	0.053	0.138	0.111	0.0704	1.02E-03
12604.05s	0.045	0.013	0.111	0.222	0.132	0.1046	/
[128] x 4	0.028	0.01	0.032	0.075	0.054	0.0398	2.44E-04
23259.82s	0.026	0.012	0.076	0.124	0.061	0.0598	/
[64] x 5	0.042	0.018	0.073	0.135	0.104	0.0744	1.02E-03
15490.03s	0.042	0.019	0.114	0.227	0.147	0.1098	/
[128] x 5	0.023	0.012	0.028	0.077	0.055	0.039	1.72E-04
29218.73s	0.022	0.016	0.06	0.12	0.064	0.0564	/

Table 6.9: ChebyKAN: Based on the implementation of CP-KAN, the inner function is Chebyshev Polynomial + Tanh. Training epochs are set to 30,000. Under [128] × 5 got the best result among all configurations. Compared with KAN, ChebyKAN has significantly lower training time and memory usage. It can set a neural network size that is almost the same as PINN. It can also achieve almost the same level of results as PINN. And for CP-KAN, as the size increases, the changes in loss and relative error show consistency: that is, the smallest loss and the smallest relative error are obtained at the current largest size.



(a) Relative concentration of CP-KAN in size $[128] \times 5$

(b) Relative concentration of FDM as reference

Figure 6.13: Relative concentration comparison of CP-KAN and FDM at t=60.

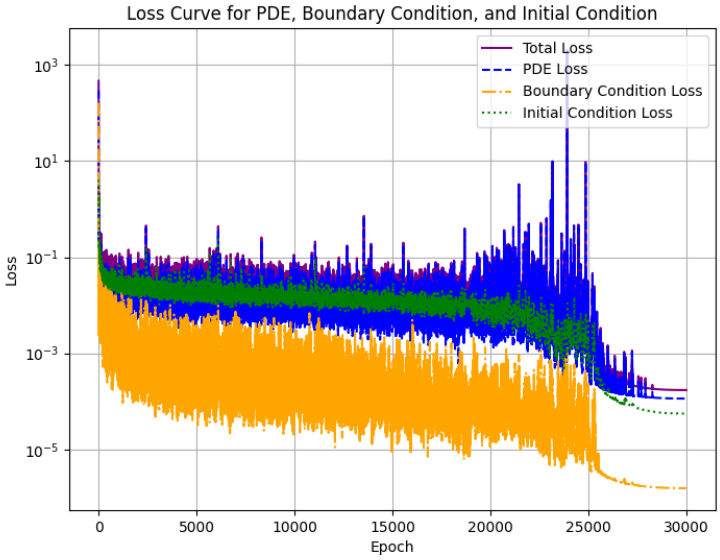


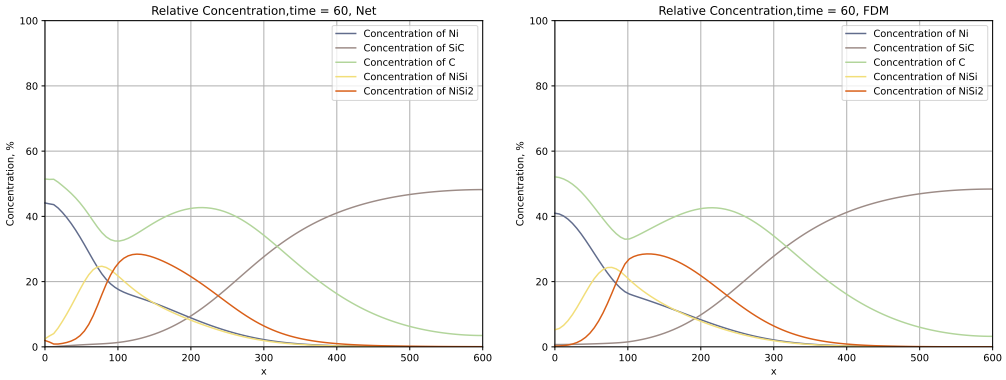
Figure 6.14: Loss curve of CP-KAN in size $[128] \times 5$

For Chebyshev-KAN, an easily noticeable difference is that the initial condition loss no longer dominates the total loss. Morever, the amplitudes of all loss oscillations, including the initial condition loss oscillations, intensify. From the loss curve, epoch 25,000 is a very clear dividing point. From here on, the loss decreases significantly, and the PDE loss gradually becomes dominant. According to the cosine annealing scheduler, this is also the stage where the learning rate decreases rapidly. Therefore, before 25,000 epochs, we can regard CP-KAN as a warm-up stage, while after 25,000 epochs, it should be an

effective learning stage. It is speculated that for CP-KAN, small learning rate + long training time is an effective training strategy.

Size	Ni	SiC	C	NiSi	NiSi2	Mean Error	Loss
[64] × 3	0.092	0.01	0.041	0.119	0.079	0.06837	2.887e-03
4266.94s	0.085	0.02	0.047	0.14	0.111	0.08047	/
[96] × 3	0.085	0.014	0.039	0.072	0.054	0.05270	3.445e-03
4125.85s	0.076	0.02	0.075	0.074	0.085	0.06589	/
[128] × 3	0.093	0.011	0.033	0.083	0.058	0.05565	2.864e-03
4296.69s	0.084	0.019	0.066	0.094	0.076	0.06814	/
[64] × 4	0.048	0.007	0.033	0.065	0.049	0.04014	2.548e-03
4482.86s	0.049	0.011	0.059	0.092	0.08	0.05824	/
[96] × 4	0.097	0.013	0.042	0.093	0.067	0.06208	1.219e-03
4676.61s	0.087	0.018	0.075	0.099	0.065	0.06876	/
[128] × 4	0.064	0.007	0.015	0.046	0.03	0.03225	2.771e-04
5360.98s	0.061	0.01	0.04	0.054	0.062	0.04536	/
[64] × 5	0.028	0.016	0.035	0.082	0.063	0.04467	3.021e-04
5196.15s	0.021	0.021	0.071	0.129	0.1	0.06844	/
[96] × 5	0.028	0.014	0.038	0.075	0.052	0.04137	3.607e-04
5983.42s	0.028	0.019	0.041	0.126	0.049	0.05257	/
[128] × 5	0.025	0.016	0.032	0.074	0.065	0.04230	1.644e-04
7214.45s	0.018	0.021	0.038	0.11	0.078	0.05286	/
[64] × 6	0.025	0.013	0.031	0.093	0.063	0.04513	4.315e-04
5988.84s	0.029	0.018	0.072	0.132	0.08	0.06653	/
[96] × 6	0.032	0.022	0.048	0.093	0.083	0.05554	9.485e-05
7140.68s	0.026	0.03	0.052	0.09	0.076	0.05463	/
[128] × 6	0.09	0.011	0.038	0.095	0.071	0.06094	2.403e-03
8153.88s	0.083	0.018	0.046	0.093	0.07	0.06348	/

Table 6.10: Attention PINN: In the case where the activation function is Tanh, training epochs are set to 30,000. Under [128] × 4 got the best result among all configurations.



(a) Relative concentration of IA-PINN in size [128] × 4

(b) Relative concentration of FDM as reference

Figure 6.15: Relative concentration comparison of IA-PINN and FDM at t=60.

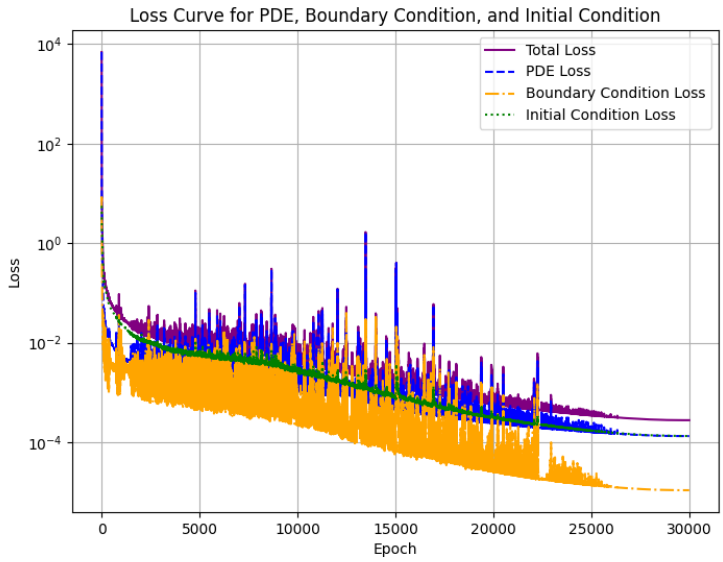


Figure 6.16: Loss curve of IA-PINN in size $[128] \times 4$

For Attention-PINN, it performs best among all models. An important feature at this time is that the initial condition loss is at the same level as the PDE condition loss. That is, the losses are in a relatively balanced state. From the relative concentration part, what is not satisfactory is that in the $x = 0$ part, Ni is higher than the benchmark, NiSi is lower than the benchmark, and $NiSi_2$ appears in a "U" shape, probably due to insufficient boundary conditions.

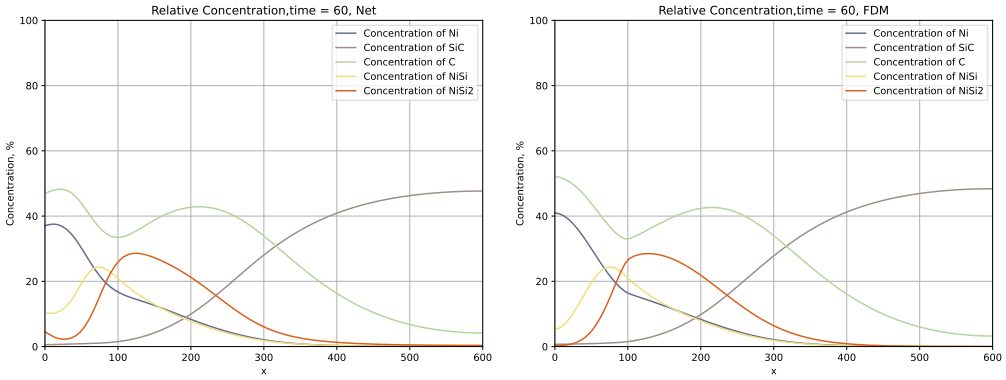
From table above, IA-PINN performances may not be good at a relatively shallow size, but as the size gradually increases, its results are much better than PINN, which effectively avoids the overfitting problem in PINN.

If oriented towards optimal results, Attention-PINN is a good choice; but if oriented towards training efficiency, the the following random Fourier features embedding method may much more suitable.

F-PINN successfully alleviates the overfitting problem faced by PINN in deep sizes, and the results are not much different at different sizes. From the table, it seems that the training time advantage of F-PINN over Attention PINN is not obvious enough, but the main result is because of the hyperparameters: the number of random Fourier features and the variance σ is a manually specified When given a smaller number of features, its training time will be similar to PINN.

Size	Ni	SiC	C	NiSi	NiSi2	Mean Error	Loss
[64] × 3	0.034	0.007	0.036	0.098	0.07	0.04887	2.157e-03
4322.34s	0.034	0.008	0.084	0.171	0.141	0.08759	/
[96] × 3	0.031	0.008	0.027	0.065	0.1	0.04607	1.851e-03
5092.64s	0.032	0.01	0.069	0.136	0.27	0.10349	/
[128] × 3	0.028	0.011	0.034	0.095	0.084	0.04976	1.242e-03
6047.83s	0.028	0.01	0.075	0.168	0.233	0.10295	/
[64] × 4	0.032	0.008	0.045	0.094	0.084	0.05254	1.455e-03
4987.56s	0.032	0.008	0.097	0.165	0.19	0.09845	/
[96] × 4	0.03	0.008	0.037	0.096	0.081	0.05040	9.571e-04
5654.12s	0.029	0.01	0.085	0.172	0.194	0.09813	/
[128] × 4	0.028	0.009	0.029	0.092	0.064	0.04433	6.019e-04
6774.73s	0.027	0.012	0.074	0.181	0.154	0.08974	/
[64] × 5	0.081	0.011	0.069	0.097	0.103	0.07234	3.015e-03
5681.87s	0.082	0.013	0.092	0.134	0.172	0.09872	/
[96] × 5	0.033	0.012	0.031	0.123	0.107	0.06124	1.596e-04
6175.62s	0.031	0.015	0.077	0.23	0.23	0.12357	/
[128] × 5	0.033	0.013	0.029	0.076	0.086	0.04769	1.660e-04
7135.89s	0.034	0.018	0.037	0.152	0.221	0.09236	/
[64] × 6	0.03	0.011	0.036	0.106	0.081	0.05278	3.379e-04
5611.90s	0.037	0.014	0.088	0.179	0.144	0.09235	/
[96] × 6	0.025	0.013	0.029	0.08	0.079	0.04527	1.445e-04
6301.60s	0.024	0.018	0.065	0.131	0.127	0.07298	/
[128] × 6	0.024	0.014	0.026	0.076	0.091	0.04632	8.856e-05
7526.34s	0.024	0.019	0.057	0.131	0.198	0.08609	/

Table 6.11: Fourier PINN: In the case where the activation function is Tanh, training epochs are set to 50,000. Add random Fourier features with the hidden layer width multiplied by 2.



(a) Relative concentration of F-PINN in size [128] × 4

(b) Relative concentration of FDM as reference

Figure 6.17: Relative concentration comparison of F-PINN and FDM at t=60.

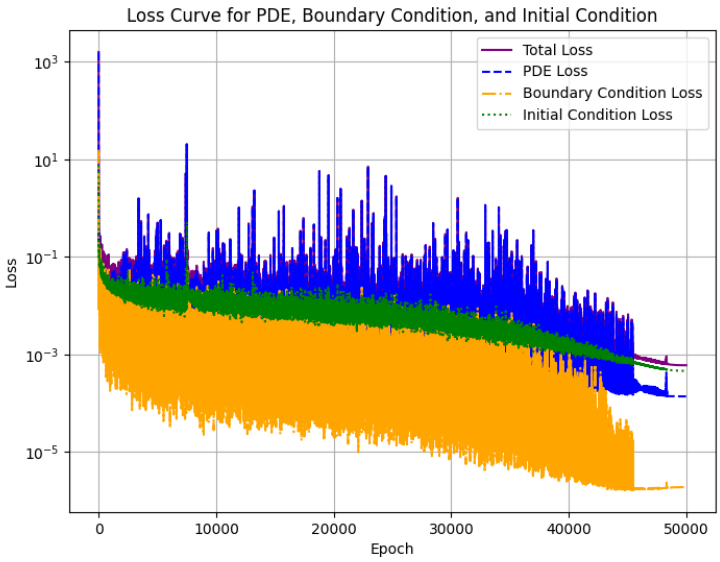


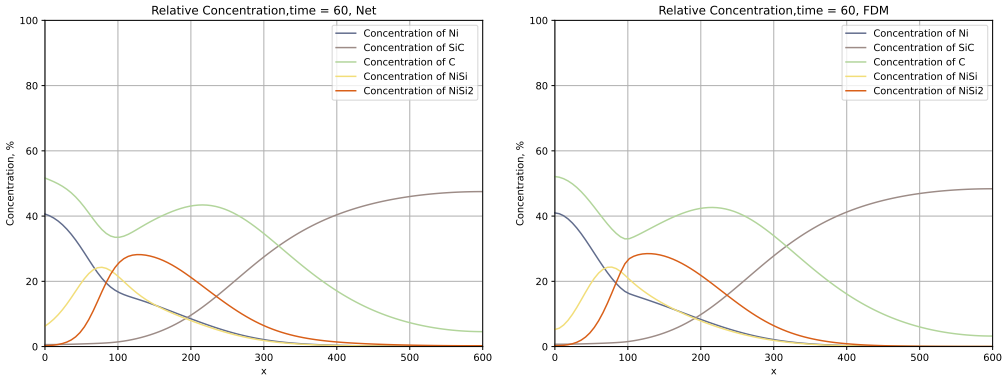
Figure 6.18: Loss curve of F-PINN in size $[128] \times 4$

Here, in the region $x \in [0, 50]$, all components perform poorly compared to the benchmark. The "U" shape is more obvious. It is speculated that the error caused by the lack of boundary conditions is magnified here. From the loss curve, we can see that the embedding of random Fourier features injects "oscillation" into the change of loss. The losses change rapidly during the training phase and converge in the later stages of training.

When the training time is sufficient, after embedding random Fourier features into the input, the attention mechanism is also introduced into the network.

Size	Ni	SiC	C	NiSi	NiSi2	Mean Error	Loss
[64] x 3	0.061	0.006	0.018	0.09	0.039	0.0428	3.91e-03
5143.59s	0.059	0.013	0.038	0.141	0.05	0.0602	/
[96] x 3	0.047	0.005	0.037	0.112	0.034	0.047	3.53e-03
5848.57s	0.049	0.009	0.081	0.234	0.052	0.085	/
[128] x 3	0.046	0.005	0.024	0.092	0.06	0.0454	2.86e-03
6627.28s	0.048	0.009	0.051	0.16	0.166	0.0868	/
[64] x 4	0.035	0.004	0.048	0.115	0.065	0.0534	2.63e-03
5774.53s	0.036	0.007	0.109	0.193	0.121	0.0932	/
[96] x 4	0.035	0.005	0.041	0.11	0.071	0.0524	1.89e-03
6621.64s	0.034	0.006	0.098	0.207	0.125	0.094	/
[128] x 4	0.034	0.005	0.031	0.099	0.07	0.0478	1.66e-03
7607.48s	0.035	0.006	0.076	0.172	0.143	0.0864	/
[64] x 5	0.031	0.005	0.057	0.068	0.029	0.038	2.16e-03
6728.26s	0.029	0.007	0.114	0.073	0.046	0.0538	/
[96] x 5	0.034	0.005	0.031	0.12	0.065	0.051	1.47e-03
7533.97s	0.035	0.006	0.076	0.204	0.091	0.0824	/
[128] x 5	0.029	0.008	0.036	0.104	0.06	0.0474	6.50e-04
8435.51s	0.029	0.011	0.088	0.169	0.116	0.0826	/
[64] x 6	0.265	0.007	0.233	0.06	0.054	0.1238	2.04e-03
8207.49s	0.555	0.009	0.58	0.062	0.063	0.2538	/
[96] x 6	0.027	0.008	0.03	0.071	0.081	0.0434	5.14e-04
8767.09s	0.026	0.01	0.076	0.114	0.098	0.0648	/
[128] x 6	0.025	0.011	0.028	0.093	0.071	0.0456	5.21e-04
9886.86s	0.024	0.014	0.067	0.182	0.187	0.0948	/

Table 6.12: AF-PINN: In the case where the activation function is Tanh, training epochs are set to 30,000. Add random Fourier features number as the hidden layer width.



(a) Relative concentration of AF-PINN in size [64] × 5

(b) Relative concentration of FDM as reference

Figure 6.19: Relative concentration comparison of AF-PINN and FDM at t=60.

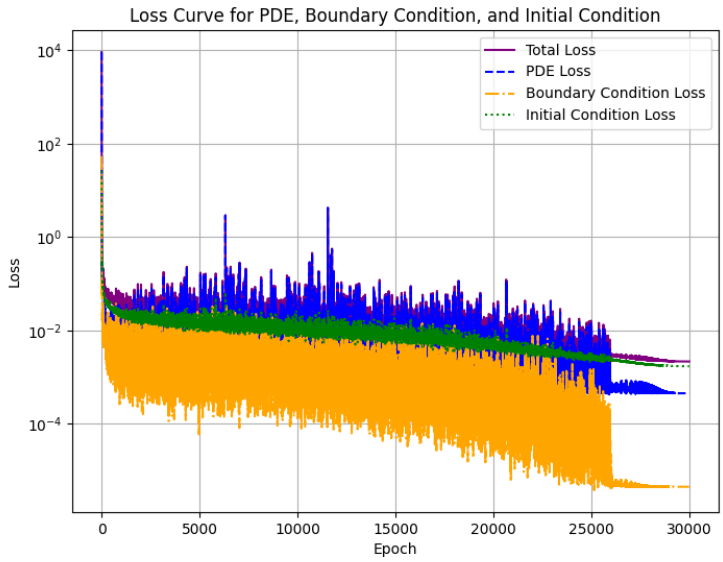
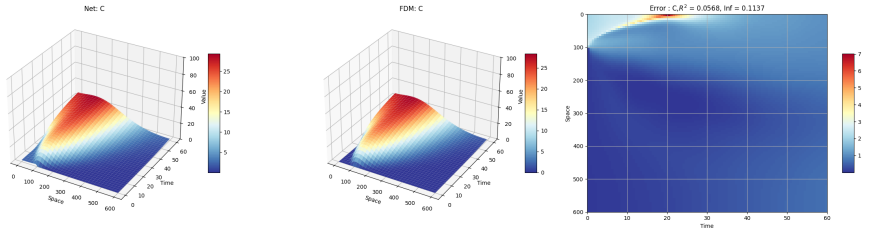


Figure 6.20: Loss curve of F-PINN in size $[64] \times 5$

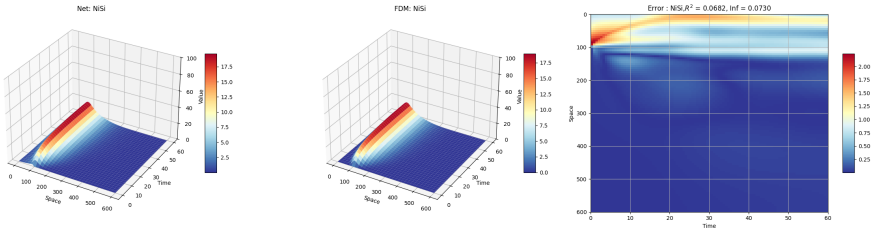
From the perspective of relative error, Attention-Fourier PINN is almost completely consistent with the FDM result at t = 60. Compared with other models, it also solves the problem well in the region $x \in [0, 50]$ where the "U" shape is prone to appear. From the loss curve, the various loss parts oscillate during the training process. In the early and middle stages, they are almost at the same level, which is a good balance between the various parts; in the later stages of training, as the learning rate decreases rapidly, the loss converges and the oscillations eases.

However, considering the entire spatiotemporal domain, the initial condition of AF-PINN at the interphase boundary, that is, $x = 100$, has relatively poor results. It is speculated that this is caused by unreasonable hyperparameter settings.

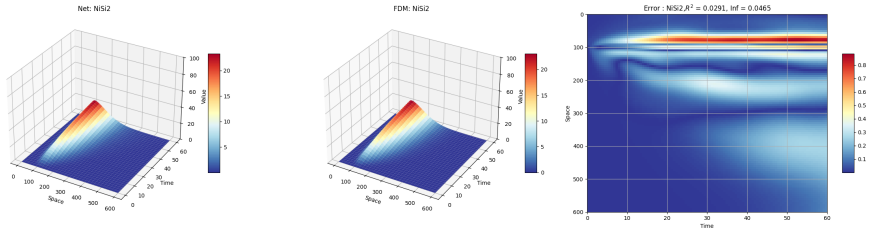
After additionally introducing L^∞ as a complement to L^2 in tables above, some situations are revealed. If the errror gap of L^2 and L^∞ is close, which indicate that the model performs more consistently across the FDM data set, with no single data point causing large errors. If the gap is significant, this means that there are a few data points with large errors, while most data points have small errors. The error distribution is not uniform and there may be a few outliers or extreme errors that make the maximum norm large.



(a) C



(b) NiSi

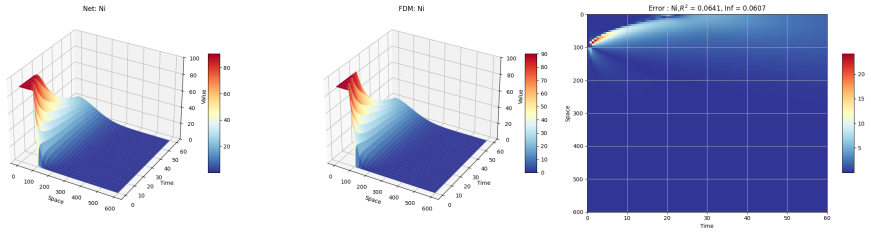


(c) $NiSi_2$

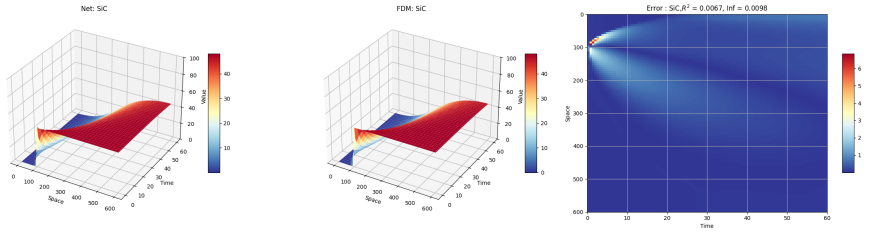
Figure 6.21: Concentration of some elements in the diffusion-reaction process. Left: Fourier Attention PINN, Right: FDM.

In the following results, I choose to use the results of Attention-PINN under $[128] \times 4$. For Ni, SiC and C, from perspective of absolute error, as discussed before, the errors are mainly concentrated at $x = 100$ and the initial part. This leads to progressively worse results as system evolves. And for NiSi and $NiSi_2$, not only the reasons mentioned above, but also the lack of boundary conditions may lead to obvious errors at $x = 0$.

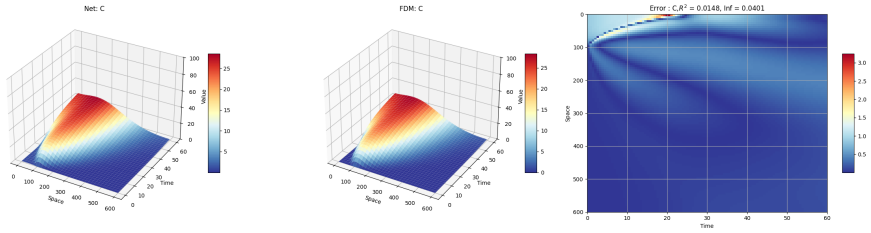
This is well shown in the absolute error graph. For most of Ni and Si, the error is kept low and the error is relatively uniform distributed. For C, NiSi, and $NiSi_2$, the error is mainly concentrated in the interphase change area, and the error distribution is uneven, so the difference between L^∞ and L^2 is obvious.



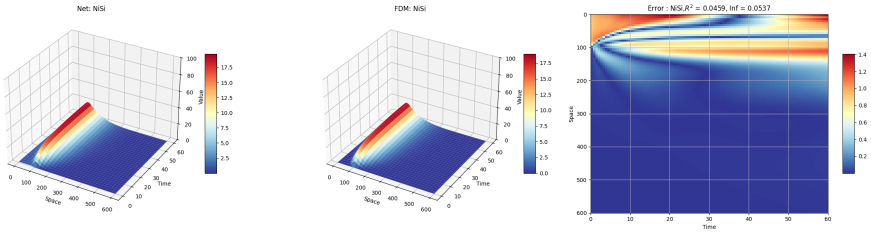
(a) Ni



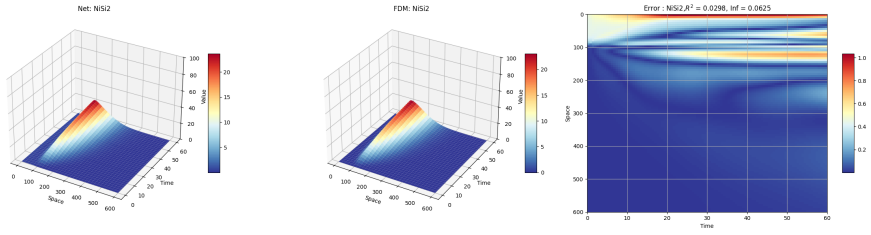
(b) SiC



(c) C



(d) NiSi



(e) $NiSi_2$

Figure 6.22: Concentration of each element in the diffusion-reaction process. Left: Attention PINN, Right: FDM. The error still mainly comes from the initial gradual accumulation.

Chapter 7

Conclusion

In order to solve the diffusion reaction model between Ni and SiC, this study introduced some numerical methods including the traditional finite difference method and the physical information neural network. Futhuremore tried to replace the multi-layer perceptron widely used in PINN with KAN and ChebyKAN, which have recently became a popular topic.

When choosing the activation function, I initially chose the silu: $f(x) = x \cdot \frac{1}{1+e^{-x}}$, which is commonly used in diffusion reactions. Because numerically it is mostly positive, which is consistent with the physical meaning of concentration. However, due to the particularity of this problem, the initial condition is a step function. It is not continuous, and in practical applications, especially in neural networks, such discontinuous and non-differentiable activation functions are usually not exist.

Talking about fitting step-like behavior with silu or tanh, we are actually discussing how these smooth, differentiable functions approximate a binary decision boundary, that is, whether they can effectively separate the input space into two categories, similar to the effect of a step function. But in fact, tanh performs better. Because it can push the neural network weights to extreme values faster and achieve strong classification effects.

In terms of improving prediction accuracy and convergence speed, several strategic PINN enhancements have been implemented and evaluated. These improvements include attention-like mechanisms that help focus on relevant features of the solution space, and random Fourier features which address the fundamental limitations of neural networks in capturing high-frequency components. These modifications have shown promising results in enhancing the overall performance of the neural network-based solver.

The main findings of this study reveal several significant insights: While KAN and ChebyKAN demonstrate superior performance in pure function fitting tasks compared to traditional MLPs, the landscape changes when dealing with complex physical problems. When considering both the accuracy of results and computational efficiency in practical

applications, the conventional MLP-based PINN architecture maintains substantial advantages, particularly in handling coupled physical phenomena like the Ni-SiC diffusion reaction system.

As thoroughly examined in the error theory section, the implementation of random Fourier feature embedding has proven to be a particularly effective method for improving the spectral bias inherent in MLP architectures. This approach has shown impressive performance across all three problems investigated in this paper, consistently outperforming the original PINN implementation in terms of accuracy and convergence speed.

Although our problem has been initially solved, there is still potential for improvement, further research is necessary to explore its space reach higher accuracy:

- The weights between the various parts of the loss are still an open question. How to balance the weights so that each part can be fully trained still has a lot of room for improvement.
- The diffusion reaction process is mainly concentrated in the interphase boundary, and other regions remain relatively stable. So rather than sampling uniformly across regions, more flexible sampling schemes or domain-decomposed parallel training might be worth exploring more deeply.
- The selection of parameters σ in random Fourier feature embedding is still based on experience, and inappropriate selection may lead to poor results. Perhaps finding a way to dynamically adjust this σ value is a reasonable solution.

Bibliography

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäSS aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Erlangen, den 29.Januar 2025 .....

Xuepeng Cheng