Group 16 Final Report

Ji Zhang, Weici Pan, Yufeng Wang, Zian Wang

May 2024

1 Introduction

Currently, computational integration of data and mechanisms, as represented by Deep Potential [5], is developing vigorously. This includes the widely recognized and rapidly developing PINN series (PINN [13], XPINN [2]), the operator learning series (DeepONet [10], Fourier Neural Operator (FNO) [8]), and some comprehensive methods like PINO [9], Physics-Informed DeepONets [4], and the code library DeepXDE [11]. Combining physical modeling and machine learning, high-performance simulation that balances physical accuracy and performance transforms our understanding of science into large-scale applicable programs and systems.

Electrochemical simulation is widely used in battery design and computation [3], representing a typical complex multi-scale problem. It involves molecular, micro, continuum, and macro scales, where multiple scales of PDEs/ODEs (partial and ordinary differential equations) are coupled, making parameter estimation difficult. The high dimensionality of equations when using finite element or finite difference methods makes the problem challenging to solve [1]. Therefore, we aim to use machine learning to simulate this complex system. As previously mentioned, the use of machine learning to solve physical problems and differential equations is rapidly developing. The key challenge is embedding physical information into the network to enhance numerical accuracy and model interpretability, ensuring the machine learning predictions are physically accurate, such as maintaining symmetry and conservation [15].

2 Formulation

Electrochemical issues are complex multi-scale problems. This section abstracts the control equations at the particle dimension.

The control equations for the particle phase are as follows:

$$\frac{\partial c_s}{\partial t} = \frac{1}{r^2} \cdot \frac{\partial}{\partial r}\left(D_s r^2 \left(\frac{\partial c_s}{\partial r}\right)\right), \quad t > 0, \quad r \in [0, r_p]. \tag{1}$$

represents an axisymmetric diffusion equation in spherical coordinates, where c_s is the lithium concentration inside the particle [mol/m^3], D_s is the **diffusion**

coefficient [m^2/s], and r_p is the size of the particle [m]. The initial condition is

$$c_s(r, 0) = c_0, \quad r \in [0, r_p].$$

We define the flux as follows

$$\boldsymbol{J}_s \equiv -D_s \nabla c_s [\text{mol}/(\text{m}^2\text{s})].$$

The boundary conditions for the diffusion equation (1) consist of two Neumann boundary conditions. At the left boundary $r = 0$, the condition is relatively trivial, specified as $r^2 D_s(\partial c_s/\partial r) = 0$. The right boundary, on the other hand, is more complex, which we express as

$$-D_s \frac{\partial c_s}{\partial r}\Big|_{r=r_p} = \frac{i_{\text{loc}}}{F}, \tag{2}$$

where F is the **Faraday constant** [C/mol], and i_{loc} is the flux of lithium intercalation/deintercalation at the particle surface [$\text{C}/(\text{m}^2\text{s})$]. It is evident that the units on the right-hand side, [$\text{mol}/(\text{m}^2\text{s})$], are consistent with the flux units on the left side. Based on the B-V (Butler-Volmer) equation, it can be derived:

$$\frac{i_{\text{loc}}}{F} = k c_s^{\alpha_c} (c_{s,m} - c_s)^{\alpha_a} c_l^{\alpha_a} \left[\exp\left(\frac{\alpha_a F \eta}{RT}\right) - \exp\left(-\frac{\alpha_c F \eta}{RT}\right) \right],$$

Here, $c_{s,m}$ represents the maximum lithium ion concentration that can be sustained within the particle. In this case, we take $\alpha_c = \alpha_a = 0.5$, and reorganize the above expression:

$$\frac{i_{\text{loc}}}{F} = 2k \sqrt{c_l c_s (c_{s,m} - c_s)} \sinh \frac{F\eta}{2RT}, \tag{3}$$

Here, $\eta = \phi_s - \phi_l - \phi_{\text{eq}}(c_s)$ is the overpotential [V]. The relationship between ϕ_{eq} and c_s is given by an empirical formula. k is a reaction rate coefficient, with units of [$\text{mol}^{-1/2}\text{m}^{5/2}\text{s}^{-1}$], primarily due to the values chosen for α_a and α_c. Although equation (1) is linear, it is a **nonlinear** diffusion equation due to its nonlinear boundary conditions, and its solution is **difficult** to obtain using Green's function or series solutions. Additionally, its boundary conditions are constantly changing over time, and traditional solvers have poor numerical stability, so we need to attempt solving it using machine learning.

Next, we simplify the equation using dimensionless techniques, normalizing certain coefficients, and make the following substitutions:

$$r = r_p \tilde{r}, \quad t = \tau \tilde{t}, \quad c = c_{s,m} \tilde{c},$$

These are then substituted into Eq. (1) to obtain:

$$\frac{\partial \tilde{c}_s}{\partial \tilde{t}} = \left(\frac{D_s \tau}{r_p^2}\right) \frac{1}{\tilde{r}^2} \frac{\partial}{\partial \tilde{r}} \left[\tilde{r}^2 \left(\frac{\partial \tilde{c}_s}{\partial \tilde{r}} \right) \right], \quad \tilde{t} > 0, \quad \tilde{r} \in [0, 1]. \tag{4}$$

Let $\tau = r_p^2/D_s$. This makes the coefficient inside the parenthesis on the right side of the equation become 1, specifically:

$$\frac{\partial \tilde{c}_s}{\partial \tilde{t}} = \frac{1}{\tilde{r}^2} \cdot \frac{\partial}{\partial \tilde{r}} \left[\tilde{r}^2 \left(\frac{\partial \tilde{c}_s}{\partial \tilde{r}} \right) \right], \quad \tilde{t} > 0, \quad \tilde{r} \in [0, 1].$$

Do the same thing to Eq. (2) and Eq. (3), and we have

$$\left(\frac{D_s c_{s,m}}{r_p} \right) \frac{\partial \tilde{c}_s}{\partial \tilde{r}} \Big|_{\tilde{r}=1} = - \left(2k c_{s,m}^{3/2} \right) \sqrt{\tilde{c}_l \tilde{c}_s (1 - \tilde{c}_s)} \sinh \frac{\tilde{\eta}}{2},$$

where $\tilde{\eta} = F\eta/(RT)$. Define $\tilde{k} = k r_p \sqrt{c_l}/D_s = k\sqrt{c_l}\tau/r_p$, and the

$$\frac{\partial \tilde{c}_s}{\partial \tilde{r}} \Big|_{\tilde{r}=1} = -2\tilde{k} \sqrt{\tilde{c}_s (1 - \tilde{c}_s)} \sinh \frac{\tilde{\phi}_s - \tilde{\phi}_l - \tilde{\phi}_{\text{eq}}(\tilde{c}_s)}{2},$$

According to dimensions, we can define another reaction time scale $\tau_{\text{react}} = r_p/(k\sqrt{c_l})$, so in fact $\tilde{k} = \tau/\tau_{\text{react}}$, where τ is the diffusion time scale $\tau_{\text{diff}} = r_p^2/D_s$. There is also an initial condition:

$$\tilde{c}_s(\tilde{r}, 0) = \tilde{c}_0, \quad \tilde{r} \in [0, 1].$$

Therefore, our task is to find $\tilde{c}_s(\tilde{r}, \tilde{t})$ under different conditions. The variables include

$$\tilde{\phi}_s - \tilde{\phi}_l, \ \tilde{k}, \ \tilde{c}_0,$$

From here on, for convenience, we will omit the tilde notation '~' unless specifically noted; below, we will handle and analyze the dimensionless quantities.

3 Fourier Neural Operator

Neural Operator. The Neural Operator was introduced in 2020 by Li et al. [7] and is designed as an iterative computation process $v_0 \mapsto v_1 \mapsto \cdots \mapsto v_T$, where v_j $(j = 0, 1, \ldots, T-1)$ are a series of functions, each taking values in $\mathbb{R}^{d_v}$. The input $a \in \mathcal{A}$ is first lifted to a higher dimensional representation $v_0(x) = P(a(x))$, typically parameterized by a shallow fully connected neural network. Multiple updates $v_t \mapsto v_{t+1}$ are then performed (as defined below). The output $u(x) = Q\left(v_T(x)\right)$ is obtained by projecting v_T through the local transformation $Q : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$. In each iteration, the update $v_t \mapsto v_{t+1}$ is defined as a combination of a non-local integral operator $\mathcal{K}$ and a local nonlinear activation function σ.

Definition 3.1 (Iterative Update) *Define the update $v_t \mapsto v_{t+1}$ as*

$$v_{t+1}(x) = \sigma\left(W v_t(x) + \left(\mathcal{K}(a; \phi) v_t\right)(x)\right), \quad \textit{for any } x \in \mathcal{D} \tag{5}$$

where $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}\left(\mathcal{U}\left(\mathcal{D}; \mathbb{R}^{d_v}\right), \mathcal{U}\left(\mathcal{D}; \mathbb{R}^{d_v}\right)\right)$ is mapped to linear operator on $\mathcal{U}\left(\mathcal{D}; \mathbb{R}^{d_v}\right)$ and is parameterized by $\phi \in \Theta_{\mathcal{K}}$, $W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ is a linear transformation $\sigma : \mathbb{R} \to \mathbb{R}$ is a point-wise acting nonlinear activation function.

We choose $\mathcal{K}(a;\phi)$ to be a kernel function integral transform parameterized by a neural network.

Definition 3.2 (Kernel Function Integral Operator $\mathcal{K}$) *Define the Kernel Function Integral Operator in Eq. (5) to be*

$$\left(\mathcal{K}(a;\phi)v_t\right)(x) = \int_{\mathcal{D}} \kappa(x,y,a(x),a(y);\phi)v_t(y)\,\mathrm{d}y, \quad \textit{for any} x \in \mathcal{D}, \tag{6}$$

where $\kappa_\phi : \mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v \times d_v}$ is a neural network parameterized by $\phi \in \Theta_{\mathcal{K}}$.

The paper on Fourier Neural Operator (Fourier Neural Operator) [8] proposes replacing the kernel function integral operator in Eq. (6) with a convolution operator defined in Fourier space. Let $\mathcal{F}$ be the Fourier transform of the function $f\colon \mathcal{D} \to \mathbb{R}^{d_v}$, and $\mathcal{F}^{-1}$ be its inverse transform. For $j = 1,\ldots,d_v$, we have

$$(\mathcal{F}f)_j(k) = \int_{\mathcal{D}} f_j(x)\mathrm{e}^{-2i\pi\langle x,k\rangle}\,\mathrm{d}x, \quad \left(\mathcal{F}^{-1}f\right)_j(x) = \int_{\mathcal{D}} f_j(k)\mathrm{e}^{2i\pi\langle x,k\rangle}\,\mathrm{d}k.$$

In Eq. (6), let $\kappa_\phi(x,y,a(x),a(y)) = \kappa_\phi(x-y)$ and use the convolution theorem, we have,,

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

thus achieving direct parameterization of κ_ϕ in Fourier space.

Definition 3.3 (Fourier Integral Operator $\mathcal{K}$) *Define the Fourier Integral Operator as*

$$\left(\mathcal{K}(\phi)v_t\right)(x) = \mathcal{F}^{-1}\left(R_\phi\cdot\left(\mathcal{F}v_t\right)\right)(x). \quad \textit{for any} x \in \mathcal{D} \tag{7}$$

where R_ϕ is the Fourier transform of a periodic function $\kappa : \bar{\mathcal{D}} \to \mathbb{R}^{d_v \times d_v}$, parameterized by $\phi \in \Theta_{\mathcal{K}}$.

For frequency modes $k \in \mathcal{D}$, we have $\left(\mathcal{F}v_t\right)(k) \in \mathbb{C}^{d_v}$ and $R_\phi(k) \in \mathbb{C}^{d_v \times d_v}$. It should be noted that since we assume κ is periodic, it has a Fourier series expansion, thus we can use discrete modes $k \in \mathbb{Z}^d$. We perform finite-dimensional parameterization by truncating the Fourier series at a finite number of modes $k_{\max}$.

$$k_{\max} = |Z_{k_{\max}}| = \left|\left\{k \in \mathbb{Z}^d : |k_j| \le k_{\max,j}, \text{for} j = 1,\ldots,d\right\}\right|,$$

where $|\cdot|$ represents the number of elements in the set. Therefore, we directly parameterize R_ϕ as a complex-valued $(k_{\max} \times d_v \times d_v)$-dimensional tensor containing a set of truncated Fourier modes, henceforth ϕ will be omitted from the notation. Since κ is real-valued, we impose conjugate symmetry. We achieve high-frequency suppression by truncating high-frequency terms as it can be efficiently implemented.

4 Explainability

For the interpretability or explainability of the model, we construct the network architecture with consideration of boundary conditions, which directly reflects the embedded physical information.

First, we revisit the Green's function solution theory. The equation to be solved is:

$$\begin{array}{ll} \mathbf{L}_{\mathbf{x},t}[u] \equiv \left(\frac{\partial}{\partial t} - \nabla^2\right) u(\mathbf{x},t) = 0, & \text{for any } \mathbf{x} \in D, \\ \frac{\partial u(\mathbf{x}_b,t)}{\partial \mathbf{n}} = g\left(\mathbf{x}_b,t\right), \quad u(\mathbf{x},0) = h(\mathbf{x}), & \text{for any } \mathbf{x}_b \in \partial D, \mathbf{x} \in D. \end{array} \tag{8}$$

Its solution is:

$$\begin{aligned} u(\mathbf{x},t) = &\int_0^t \mathrm{d}\tau \int_D \mathrm{d}^m\mathbf{y}\, G(\mathbf{x},\mathbf{y};t-\tau) f(\mathbf{y},\tau) + \int_D u(\mathbf{y},0) G(\mathbf{x},\mathbf{y};t)\, \mathrm{d}^m\mathbf{y} \\ &+ \int_0^t \mathrm{d}\tau \int_{\partial D} \frac{u\left(\mathbf{y}_b,\tau\right)}{\partial \mathbf{n}_y} G\left(\mathbf{x},\mathbf{y}_b;t-\tau\right)\, \mathrm{d}S \end{aligned} \tag{9}$$

The initial version of the Fourier Neural Operator, when considering the solution operator of a differential equation, only considered the function's value, without incorporating boundary conditions as in Eq. (9). That is to say, it could only handle situations with fixed boundary conditions and could not learn the solution operator for differential equation systems with changing boundary conditions. The equation we aim to solve is a diffusion equation with continually changing boundary conditions, which cannot be resolved with existing operator learning techniques. Hence, we proposed an operator learning that includes boundary conditions, starting from Eq. (9), and we designed an algorithm as shown in Fig. 1 .

We implement a fixed-time-step algorithm where the inputs at time t are $u(x)$ and $bc(x)$, where $bc(x)$ represents the function of boundary conditions on x (same below), and the output is $u_{t+\Delta t}(x)$ at time $t + \Delta t$. Compared to the original FNO algorithm, it includes the following modules: 1. Preprocessing and feature extraction module for boundary conditions; 2. Fourier convolution module shared by boundary condition features and function value features; 3. Feature fusion module in the Fourier layer. These will be introduced in turn.

4.1 Preprocessing and Feature Extraction Module for Boundary Conditions

We consider a regular grid region D and its boundary ∂D. At fixed time t, $u(x)$ is a tensor of dimension $s_1 \times \cdots \times s_d$, and we also pad the boundary conditions $bc(x)$ with zeros to match the same tensor dimension, which only has values on the boundary. This is to ensure that it can share the same convolution parameters in the Fourier layer with the features of $u(x)$. After zero-padding, we perform a dimension-invariant linear transformation on each element of $bc(x)$ to ensure it is on the same scale as $u(x)$, thus sharing the same $Lift$ and $Proj$ layers for transformations between high-dimensional hidden spaces and the original space.

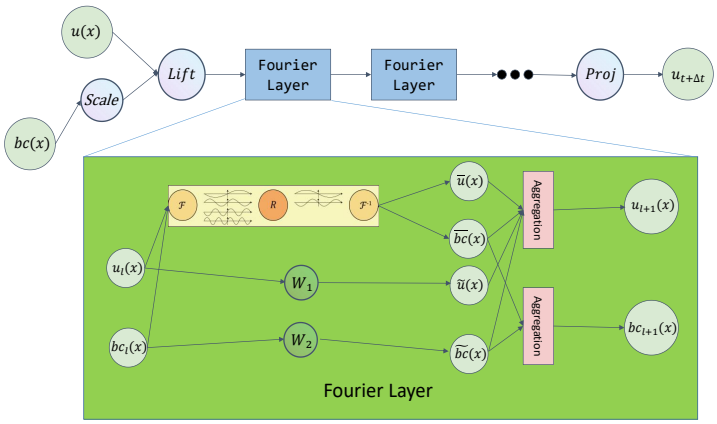

Figure 1: Neural Operator Network Architecture Including Boundary Conditions

4.2 Fourier Convolution Module Shared by Boundary Condition Features and Function Value Features

According to Eq. (9), the boundary condition $u\left(\mathbf{y}_b, \tau\right) / \partial \mathbf{n}_{\mathbf{y}}$ and $f(\mathbf{y}, \tau)$ share the same convolution function. Since we have padded $u\left(\mathbf{y}_b, \tau\right) / \partial \mathbf{n}_{\mathbf{y}}$ throughout region D, we can transform the representations of $u(x)$ and $bc(x)$ in Fourier space using the same linear transformation parameters R.

4.3 Feature Fusion Module in the Fourier Layer

In each Fourier layer, $u_l(x)$ and $bc_l(x)$ go through the Fourier convolution module to obtain $\bar{u}(x)$ and $\bar{bc}(x)$, and also pass through two linear layers separately, obtaining $\widetilde{u}(x)$ and $\widetilde{bc}(x)$. Then, $\bar{u}(x)$, $\bar{bc}(x)$, $\widetilde{u}(x)$, and $\widetilde{bc}(x)$ undergo a linear combination to become $u_{l+1}(x)$; and $\widetilde{bc}(x)$ and $\bar{bc}(x)$ undergo a linear combination to become $bc_{l+1}(x)$.

5 Results

5.1 Solving the Fixed Constant Boundary Diffusion Model

Under fixed boundary conditions, we compare the results generated by traditional numerical solvers with those of the original FNO and the boundary-condition-aware FNO. Our experimental setup is as follows: the optimizer is Adam, the optimization strategy is CosineAnnealingLR, the dataset size covers the first 90 time steps for training and the last 10 time steps for testing, the initial learning rate is 5×10^{-4}, the loss function is mean squared error, and bc

represents boundary conditions. The mean relative error results on the test set are shown in Tbl. 1.

Table 1: Comparison of Fixed Constant Boundary Effects (bold indicates better performance)

Networks	$bc = 0.01$	$bc = 0.02$	$bc = 0.03$	$bc = 0.04$
FNO-BC	$\mathbf{2 \times 10^{-5}}$	$\mathbf{9 \times 10^{-5}}$	$\mathbf{1.1 \times 10^{-4}}$	2.8×10^{-4}
FNO	9×10^{-5}	1.8×10^{-4}	2.7×10^{-4}	$\mathbf{2.2 \times 10^{-4}}$

Results in the table indicate that, even in scenarios with fixed constant boundaries where no additional boundary conditions are needed, incorporating boundary conditions can potentially enhance the model's performance. This might be because the boundary condition branch can globally adjust the magnitude of the final output values, thereby making the model more robust.

5.2 Solving the Mixed Constant Boundary Diffusion Model

In this scenario, we first generate simulation data with fixed constant boundaries using traditional numerical solvers. Then, we conduct mixed training and testing with data where boundary conditions are set to 0.01, 0.02, 0.03 and 0.04. We find that the original FNO is unable to handle this situation, whereas the boundary-condition-aware FNO performs significantly better. The mean squared error on the test set is shown in Tbl. 2.

Table 2: Comparison of Mixed Constant Boundary Effects

Networks	Mixed Boundaries
FNO-BC	$\mathbf{1.2 \times 10^{-4}}$
FNO	3.6×10^{-4}

5.3 Analysis of the Method's Problems

We have found that this algorithm is highly sensitive to hyperparameters. Unlike traditional tasks [12], the loss surface during training on the test set is rather pathological for this algorithm. As shown in Fig. 2 , with learning rates in the legend from top to bottom being 0.0005, 0.0005, 0.01, 0.0004, and 0.0002, it is evident that even with the same or similar learning rates, the model's performance during training can vary significantly, leading to large discrepancies in training outcomes. From the results of failed training sessions, the model is very prone to falling into poor local minima, making optimization challenging. Additionally, since the problem we are solving is a linear differential equation, it must adhere to the properties expected of linear differential equations, and cannot arbitrarily use some classical techniques that make the neural network's loss surface smoother, such as batch normalization [14]. Therefore, future work will

need to analyze this algorithm according to optimization theory and approximation theory [6], adopting correct initialization, regularization, and optimizers to improve optimization stability and enhance its generalization capabilities across datasets with larger data ranges.

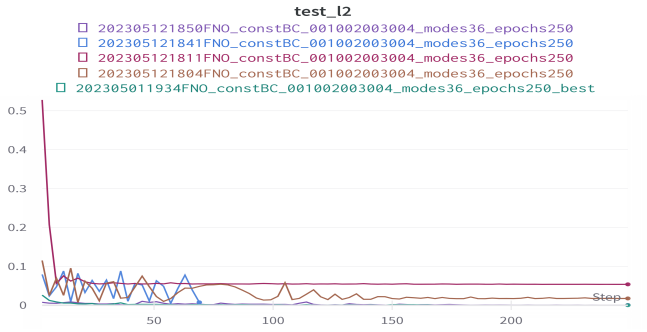

Figure 2: Mean Squared Loss on the Test Set During Training

6 Integration with LLMs

Since the reader might be unfamiliar with the topic of this work, we use GPT-4 to help with clarifying the backgrounds, the math, and applications. Moreover, we also get answers about the value of our work when stronger LLMs come out in the future. **The following parts are generated by LLM.**

6.1 Background and Applications

What is the Research About? Your research involves combining two powerful tools: physical models and machine learning. Physical models are equations used to describe natural processes, like how batteries work or how pollution spreads in the air. Machine learning, on the other hand, is a type of computer science that allows computers to learn from data and make predictions.

Why Combine These Tools? Physical models are great for understanding detailed processes based on scientific principles, but they can be very complex and hard to solve, especially when dealing with big problems like battery design. Machine learning can handle large amounts of data and find patterns that humans might not see, but it often acts like a "black box," making it hard to understand how it comes to its conclusions. By combining physical models with machine learning, your research aims to tackle complex problems more effectively. This approach helps ensure that the machine learning predictions are not only accurate but also make sense in terms of physical laws (like conserving energy).

Specific Focus: Fourier Neural Operator (FNO) A key part of your research is improving a tool called the Fourier Neural Operator (FNO), which

is a special kind of machine learning model designed to work with the types of equations used in physical models. You're working on making FNO better at handling boundary conditions, which are the conditions set at the edges of the area where the problem is being solved (imagine setting the temperature at the boundary of a heated plate).

Applications: Battery Design One practical application of your research is in designing better batteries. Batteries involve many different processes happening at different scales, from tiny chemical reactions inside the battery to the overall behavior of the entire battery pack. Your improvements to machine learning models help simulate these processes more accurately, which can lead to designing batteries that are more efficient, last longer, and are safer.

6.2 Explain the Math

Operator Learning Operator learning is a technique that learns relationships between sets of input and output functions. This is particularly suited for problems where both the inputs and outputs are continuous functions, common in many scientific and engineering applications. **Function Mapping:** At its core, operator learning involves creating a model that maps input functions to output functions, essentially learning a rule or set of rules that can predict outputs given new inputs. **Parameter Optimization:** It uses parameters within its model structure that are continuously adjusted to minimize errors between the predicted outputs and actual outputs, enhancing the model's accuracy over time.

Fourier Neural Operators (FNO) Fourier Neural Operators are designed to solve partial differential equations (PDEs) efficiently by leveraging the mathematical properties of Fourier transforms. **Fourier Transform:** FNOs apply the Fourier transform to convert functions into a frequency space where convolution operations (a type of mathematical manipulation) are simpler and more computationally efficient. **Convolution in Frequency Space:** By performing convolution in the Fourier space, FNOs simplify the interactions in the data, which are typically complex in the time or spatial domain. This approach effectively filters and processes the information to solve PDEs rapidly.

Green's Function Solution Theory Green's functions provide a method to solve differential equations that describe physical phenomena. They help predict how systems respond to external stimuli or initial conditions at any given point. **Response to Impulses:** A Green's function calculates the system's response at every point when subjected to an impulse at any other point. This fundamental property is used to construct the solution to a differential equation. **Superposition Principle:** By integrating these responses (using the Green's functions), you can predict the overall behavior of a system under a range of forces or initial conditions.

6.3 Value with stronger LLMs in the future

Advanced Integration of AI and Physical Modeling As LLMs grow more capable, they could better integrate the complex mathematical models used in your research with AI-driven systems. This could lead to more sophisticated simulations that accurately represent the physical world, increasing the reliability and utility of models in practical applications like energy systems and environmental management.

Improved Problem Solving and Optimization Stronger LLMs could enable more efficient problem-solving capabilities, particularly in how they handle the optimization of systems described by your research. This includes optimizing battery life in energy applications or improving efficiency in computational tasks that are currently limited by the complexity of the underlying mathematical problems.

Enhanced Predictive Analytics With enhanced capabilities, LLMs could offer more accurate and far-reaching predictive analytics. This would be particularly valuable in fields that rely heavily on accurate predictions of complex systems, such as weather forecasting, climate modeling, and advanced materials design. The ability to predict and model scenarios with higher accuracy would be a substantial advancement.

Broader Applicability Across Industries As LLMs evolve, the techniques developed in your project could be applied more broadly and deeply across various industries. This might include sectors not traditionally associated with advanced mathematical modeling, such as healthcare for predictive diagnostics, finance for risk assessment, and even urban planning for infrastructure development.

Accelerated Innovation and Research Stronger LLMs could accelerate the pace of research and development in computational mathematics and physical sciences. By automating some of the more complex aspects of research, such as the tuning of models and the interpretation of vast datasets, LLMs could free researchers to focus on innovation and theory development.

Educational and Training Tools With their advanced understanding and processing capabilities, LLMs could be used to develop more effective educational tools that help students and professionals better understand and apply the principles of your research. This could democratize access to advanced scientific knowledge, making it easier for people to learn about complex systems and their applications.

References

[1] Raul Ciria Aylagas, Clara Ganuza, Ruben Parra, Maria Yañez, and Elixabete Ayerbe. Cidemod: An open source tool for battery cell inhomogeneous performance understanding. *Journal of the Electrochemical Society*, 169(9):090528, 2022.

[2] Ameya D. Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

[3] Marc Doyle, Thomas F. Fuller, and John Newman. Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *Journal of the Electrochemical Society*, 140(6):1526, 1993.

[4] Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed deep neural operator networks. *arXiv preprint arXiv:2207.05748*, 2022.

[5] Jiequn Han, Linfeng Zhang, Roberto Car, and Weinan E. Deep potential: A general representation of a many-body potential energy surface. *Communications in Computational Physics*, 23(3), 2018.

[6] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *The Journal of Machine Learning Research*, 22(1), 2021.

[7] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[8] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2021.

[9] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2023.

[10] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[11] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[12] James Lucas, Juhan Bae, Michael R. Zhang, Stanislav Fort, Richard Zemel, and Roger Grosse. Analyzing monotonic linear interpolation in neural network loss landscapes. *arXiv preprint arXiv:2104.11044*, 2021.

[13] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[14] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *arXiv preprint arXiv:1805.11604*, 2019.

[15] Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. *arXiv preprint arXiv:2002.03061*, abs/2002.03061, 2020.