

TDK-dolgozat

Laczkó Csongor Loránd



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

A transferable PINN-based method for quantum graphs with unseen structure

Csongor Loránd Laczkó
Computer Science Engineering BSc

2024

Supervisors: Prof. Mihály Kovács, Mihály András Vághy

Abstract

This study introduces a transferable approach for solving partial differential equations (PDEs) on metric graphs, often called quantum graphs, employing Physics-Informed Neural Networks (PINNs). Unlike traditional solvers constrained by specific graph structures, our method utilizes a Neumann-Neumann domain decomposition technique, offering flexibility across various network topologies. This approach aims to provide improved adaptability to previously unseen graph structures, potentially broadening the applicability of PDE solutions to diverse network configurations.

1 Introduction

Quantum graphs, mathematical abstractions representing networks of quasi-one-dimensional quantum systems coupled at vertices, have emerged as powerful tools for studying a diverse array of physical phenomena, spanning from superconductivity in granular materials [2], wave guide networks [18, 19], cell differentiation [14] and quantum transport processes [37]. These structures, often referred to as *quantum wires*, provide a versatile framework for modelling complex quantum systems with varying structural arrangements. The interdisciplinary nature of quantum graphs has led to their applications in condensed matter physics [21, 23], quantum chemistry [3], and mathematical physics [1], solidifying their position as an indispensable tool for understanding quantum systems. For further applications, we refer to [6].

In recent years, there has been a growing interest in developing efficient numerical techniques for solving quantum graph problems, especially in scenarios where the structure of the graph is not entirely known a priori or when its size renders traditional approaches obsolete through memory requirements. This suggests the use of domain decomposition methods. While these methods have rarely been used for quantum graphs, an overlapping decomposition on domains was introduced in [36] more than 150 years ago, further developed in [5, 33, 40]. Then nonoverlapping decompositions were introduced due to their inherent parallelism and the growth of high-performance computing (HPC) [17, 27, 28]. We use a Neumann-Neumann method, originally introduced in [9, 10, 16, 42], and generalized for quantum graphs in [25]. For surveys on domain decomposition methods, we refer to [12, 47]. A more thorough theoretical background and historical overview can be found in [32, 38, 43].

Relying on recent advances in machine learning and the robust increase of computational power, several modern numerical approaches have been developed for evolution equations that approximate or enhance traditional solvers [34, 24, 11, 29]. In particular, Physics-Informed Neural Networks (PINNs) have been designed to solve complex physical problems by integrating prior knowledge into the learning process [34]. PINNs extend the applicability of neural networks to nonlinear problems without the need for linearization or other restrictive assumptions. Automatic differentiation, inherent in the functioning of neural networks, plays an important role in this method. By differentiating neural networks with respect to their input coordinates, physics-informed neural networks allow for the evaluation of their adherence to the governing equations, a process more stable numerically than finite differences. The applications of PINNs range from data-driven approaches for model inversion [13, 20, 39], through system identification [41], to the development of new classes of numerical solvers for partial differential equations [48].

While there exist PINN solutions for quantum graphs [50], they heavily depend on the structure of the graph, requiring the training of a PINN for each edge. This means that if the graph structure changes, several or all of the models may need retraining. This serves as motivation for our approach, which is designed to be applicable even for unseen graph structures. Building upon our recent submission on this topic [15], this study further advances the methodology by relaxing structural constraints and enhancing solution robustness.

The key reference for our approach is [45], where a PINN-based numerical solver for some classical PDEs on domains using domain decomposition methods was introduced. The authors trained an appropriate PINN on a single 1×1 square with Dirichlet boundary condition and used it to solve the Laplace equation and the Navier-Stokes equation on various unseen domains decomposed with an overlapping Schwarz method.

The problem we aim to address is the formulation and solution of the time-independent Schrödinger equation on quantum graphs. Let $G = (V, E)$ denote an arbitrary quantum graph; that is, a metric graph G equipped with an elliptic differential operator on each edge and certain standard vertex conditions, where V is a finite set of vertices and E represents the edges connecting these vertices. Each edge $e \in E$ is assigned a length $\ell_e \in (0, \infty)$ and a local coordinate $x \in [0, \ell_e]$. A function u on a metric graph G can be defined as a vector of functions and we write

$$u = (u_e)_{e \in E}$$

and consider it to be an element of the product function space (to be specified later). Let $u_e(v)$ denote the value of u at v along the edge e .

On a given edge e , the time-independent Schrödinger equation is expressed as follows:

$$-(c_e u'_e)'(x) + v_e(x)u_e(x) = f_e(x), \quad x \in (0, \ell_e), \quad e \in E. \quad (1)$$

In addition, we ensure the continuity of the function values along each edge at the incident vertices, while the Neumann-Kirchhoff condition is satisfied at each vertex. This condition is formulated as:

$$\sum_{e \in E_v} c_e(v) \frac{du_e(v)}{dx} = 0 \quad \forall v \in V.$$

where E_v denotes the subset of E comprising the edges incident to the vertex v . A more detailed formalization of the problem will be provided in Section 2.1.

Equation (1) embodies the fundamental principles of the time-independent Schrödinger equation, a cornerstone of quantum mechanics that governs the behavior of quantum systems. In its current formulation, this equation describes the stationary state of a quantum system, represented by the function $u_e(x)$, which corresponds to the wave function associated with the particle or system under consideration.

The coefficient $c_e(x)$ is assumed to be a positive Lipschitz continuous function. It is plausible that $c_e(x)$ may not be differentiable; hence, we transform our equation in a weak form, see Section 2.1 for more details. We assume that $v_e(x)$ is essentially a bounded function for which $v_e(x) \geq v_0$ for some $v_0 > 0$, and that f_e is finite square integrable. It is worth emphasizing the role of the Neumann-Kirchhoff conditions and the alignment of function values along every edge at incident vertices. For the sake of simplicity, in this paper we assume that $c_e(x) \equiv 1$ and that v_e is constant but can be different across edges.

The paper is structured as follows. In Section 2, we present the necessary preliminaries, including discussions on quantum graphs, the Finite Element Method (FEM) for quantum graphs, the Neumann-Neumann domain decomposition method (DDM) for quantum graphs and Physics-Informed Neural Networks (PINNs). Section 3 details the implementation aspects of our research, covering an overview of our methodology, training data generation, model architecture, and training procedures. We also discuss hyperparameter tuning and computational considerations. In Section 4, we present the results of numerical experiments conducted to validate our approach. This includes assessments of a 1D model and experiments on graph-based models. Finally, Section 5 concludes the paper with a summary of findings and potential avenues for future research.

2 Preliminaries

2.1 Quantum Graphs

In this section, we will introduce the fundamental concepts and properties associated with quantum graphs as discussed in [6].

A graph G consists of a finite or countably infinite set of vertices V and a set E of edges connecting them. Two vertices v_1 and v_2 are called adjacent (denoted by $v_1 \sim v_2$) if there exists an edge connecting them. A graph G is fully specified by its $|V| \times |V|$ adjacency matrix A . A simple graph is defined as a graph that does not contain multiple edges between any pair of vertices and does not have any loops. If we assume the graph is simple, the elements of the adjacency matrix are given by

$$A_{v_1, v_2} = \begin{cases} 1 & \text{if } v_1 \sim v_2, \\ 0 & \text{otherwise.} \end{cases}$$

A graph is called a directed graph or digraph if each of its edges is assigned a direction. In other words, each edge has one origin and one terminal vertex.

This perspective is standard in graph theory and combinatorics, where edges are considered as relations between vertices, rather than any physical or geometrical links. Thus, the term discrete or combinatorial graph is used to emphasize this. However, we will consider another perspective where the graphs are viewed as one-dimensional complexes, the edges being one-dimensional segments or *physical wires*. Such graphs will be equipped with additional structures that make them metric or quantum graphs.

Definition 2.1. *A graph G is said to be a **metric** graph if*

- (1) *each edge e is assigned a positive length $\ell_e \in (0, \infty)$*
- (2) *a coordinate $x_e \in \ell_e$ is increasing in the direction of the edge and is assigned on each edge.*

It is often important how the lengths of different edges are related. In some topics (e.g., quantum chaos or inverse problems), it helps when the lengths are as unrelated as possible, more precisely, are rationally independent. In some other instances, for example, when trying to relate spectra of quantum and discrete graphs, the opposite happens, and one benefits from assuming that all lengths are the same. This justifies the following notion:

Definition 2.2. *A metric graph G is said to be **equilateral** if the lengths of all its edges are equal: $\ell_e \equiv \ell$.*

Furthermore, most of the models we work with allow for the rescaling of edge lengths and their associated differential equations without loss of generality. This implies that we can convert the graph into an equilateral configuration, thereby ensuring uniform edge lengths throughout the entire graph. Consequently, we will proceed with the assumption that each edge has a length of $\ell = 1$.

One should notice that the points of a metric graph G are not only its vertices but also all intermediate points x on the edges. Thus, when we speak about functions $f(x)$ on G , we consider them as defined along the edges (rather than at the vertices only, as in discrete models).

This view transforms a metric graph into a quantum graph upon the assignment of a differential operator on G , commonly referred to as the **Hamiltonian**. Such a transition is accompanied by appropriate vertex conditions, facilitating the formulation of quantum graph problems.

First, we introduce function spaces on a metric graph G and differential operators defined on these spaces as outlined in [6], then proceed to explain the necessary vertex conditions.

Definition 2.3. (1) The space $L^2(\mathbf{G})$ on $\mathbf{G}$ consists of functions that are measurable and square-integrable on each edge $\mathbf{e}$ with norm

$$\|f\|_{L^2(\mathbf{G})}^2 := \sum_{\mathbf{e} \in \mathbf{E}} \|f\|_{L^2(\mathbf{e})}^2,$$

meaning that $L^2(\mathbf{G})$ is the orthogonal direct sum of spaces $L^2(\mathbf{e})$.

(2) The Sobolev space $H^1(\mathbf{G})$ consists of functions on $\mathbf{G}$ that belong to $H^1(\mathbf{e})$ for each edge $\mathbf{e}$ and such that

$$\|f\|_{H^1(\mathbf{G})}^2 := \sum_{\mathbf{e} \in \mathbf{E}} \|f\|_{H^1(\mathbf{e})}^2 < \infty.$$

Here the $H^1(\mathbf{e})$ space contains $L^2(\mathbf{e})$ functions whose first generalized derivative is also in $L^2(\mathbf{e})$, and the $H^1(\mathbf{e})$ norm is defined as

$$\|f\|_{H^1(\mathbf{e})}^2 = \|f\|_{L^2(\mathbf{e})}^2 + \|f'\|_{L^2(\mathbf{e})}^2.$$

(3) The Sobolev space $H^2(\mathbf{G})$ consists of functions on $\mathbf{G}$ that belong to $H^2(\mathbf{e})$ for each edge $\mathbf{e}$ and satisfy

$$\|f\|_{H^2(\mathbf{G})}^2 := \sum_{\mathbf{e} \in \mathbf{E}} \|f\|_{H^2(\mathbf{e})}^2 < \infty.$$

Here, the space $H^2(\mathbf{e})$ contains functions in $L^2(\mathbf{e})$ whose first and second generalized derivatives are also in $L^2(\mathbf{e})$. The $H^2(\mathbf{e})$ norm is given by

$$\|f\|_{H^2(\mathbf{e})}^2 = \|f\|_{L^2(\mathbf{e})}^2 + \|f'\|_{L^2(\mathbf{e})}^2 + \|f''\|_{L^2(\mathbf{e})}^2.$$

In addition to function continuity on $\mathbf{G}$, we require the so-called Neumann-Kirchhoff conditions to be satisfied at each vertex. This condition is expressed as:

$$\sum_{\mathbf{e} \in \mathbf{E}_v} c_{\mathbf{e}}(v) \frac{du_{\mathbf{e}}(v)}{dx} = 0 \quad \forall v \in \mathbf{V}.$$

Here, the derivatives are assumed to be taken in the directions away from the vertex, into the edge, commonly referred to as the outgoing directions, which we consistently assume throughout this work.

Continuity at degree-one vertices is trivially satisfied, and the Neumann-Kirchhoff condition simplifies to a straightforward Neumann condition with the value of 0. This implies that the derivative of the function along the edge at such vertices must vanish, aligning with the natural behaviour of one-dimensional functions.

We observe that the Neumann-Kirchhoff conditions are the natural boundary conditions satisfied by the following one-dimensional Schrödinger-type operator:

$$\mathcal{H} : u(x) \mapsto -\frac{d}{dx} \left(c(x) \frac{d}{dx} u(x) \right) + v(x)u(x).$$

By equipping a metric graph with a differential operator $\mathcal{H}$ satisfying both continuity and Neumann-Kirchhoff conditions, we attain a quantum graph.

To write the vertex conditions more compactly, let us define the vector of function values at $v \in \mathbf{V}$ as

$$U(v) = (u_{\mathbf{e}}(v))_{\mathbf{e} \in \mathbf{E}_v} \in \mathbb{R}^{d_v}$$

and the bi-diagonal matrix

$$I_v = \begin{bmatrix} 1 & -1 & & \\ & \ddots & \ddots & \\ & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{(d_v-1) \times d_v}.$$

Then $I_v U(\mathbf{v}) = 0 \in \mathbb{R}^{d_v-1}$ implies that the function values along the edges in E_v coincide at $\mathbf{v} \in V$. Similarly, we define

$$U'(\mathbf{v}) = (u'_e(\mathbf{v}))_{e \in E_v} \in \mathbb{R}^{d_v},$$

the vector of function derivatives at $\mathbf{v} \in V$ and the row vector

$$C(\mathbf{v})^\top = (c_e(\mathbf{v}))_{e \in E_v}^\top \in \mathbb{R}^{1 \times d_v}.$$

Then $C(\mathbf{v})^\top U'(\mathbf{v}) = 0$ implies that the function u satisfies the Neumann-Kirchhoff conditions at $\mathbf{v} \in V$.

Let us denote by $d_v = |E_v|$ the degree of $\mathbf{v} \in V$. We denote by $\text{int}(G)$ the set of vertices with degree $d_v > 1$ and by ∂G the set $V \setminus \text{int}(G)$. Then the quantum graphs can be formally written as

$$\begin{aligned} -(c_e u'_e)'(x) + v_e(x) u_e(x) &= f_e(x), \quad x \in (0, \ell_e), \quad e \in E, \\ 0 &= I_v U(\mathbf{v}), \quad \mathbf{v} \in \text{int}(G), \\ 0 &= C(\mathbf{v})^\top U'(\mathbf{v}), \quad \mathbf{v} \in V, \end{aligned}$$

where we seek $u_e \in H^2(G)$.

While the problem is well-posed [8, Proposition 3.1], it is convenient to introduce a weak formulation of (1) for our purposes. The corresponding bilinear form $\mathfrak{a} : H^1(G) \times H^1(G) \mapsto \mathbb{R}$ is defined as

$$\begin{aligned} \mathfrak{a}(u, \phi) &= \sum_{e \in E} \left(\int_e c_e(x) u'_e(x) \phi'_e(x) dx + \int_e v_e(x) u_e(x) \phi_e(x) dx \right), \\ D(\mathfrak{a}) &= \{u \in H^1(G) \times H^1(G) : u \text{ is continuous at every vertex}\}. \end{aligned}$$

Thus, we seek a solution $u \in D(\mathfrak{a})$ such that

$$\mathfrak{a}(u, v) = f(v), \quad \forall v \in D(\mathfrak{a}),$$

where $f(v) := \langle f, v \rangle_{L^2(G)}$. This weak formulation is also well-posed [8], and the unique solution of it is the unique solution of (1).

Utilizing weak formulations in $H^1(G)$ relaxes solution smoothness requirements, broadening the range of admissible functions. Moreover, the continuity requirement in $D(\mathfrak{a})$ ensures that any function f from this space assumes the same value at a vertex $\mathbf{v}$ on all edges adjacent to $\mathbf{v}$, thus uniquely defining $f(\mathbf{v})$ and aligning with the natural behaviour of one-dimensional functions.

2.2 Finite Element Method for quantum graphs

Finite Element Method (FEM) is a general numerical method for solving partial differential equations in one or more space dimensions. For solving a problem, the FEM breaks down the large problem into smaller, easier-to-handle pieces called *finite elements*. This is achieved by a particular space discretization in the space dimensions. The formulation of a boundary value problem results in a system of algebraic equations, which can be written in a matrix form.

FEM has been formalized for quantum graphs as well, as introduced by Arioli et al. [4]. On each edge of the quantum graph, we can use the classical one-dimensional FEM. Let us take an $\mathbf{e}$ edge identified by two vertices, $\mathbf{v}_a$ and $\mathbf{v}_b$. First, we subdivide the edge into n_e intervals of length h_e with $n_e \geq 2$. We will get points

$$\{\mathbf{V}_e = \{x_j^e\}_{j=1}^{n_e-1}\} \cup \{\mathbf{v}_a^e\} \cup \{\mathbf{v}_b^e\},$$

which form a chain linking vertex $\mathbf{v}_a^e$ to vertex $\mathbf{v}_b^e$ on the edge $\mathbf{e}$. The internal points x_j^e are called the *nodes* of the discretization. For each internal node, we denote by $\{\Psi_j^e\}_{j=1}^{n_e-1}$ the standard hat basis functions

$$\Psi_j^e(x^e) = \begin{cases} 1 - \frac{|x_j^e - x^e|}{h_e} & \text{if } x_{j-1}^e \leq x^e \leq x_{j+1}^e, \\ 0 & \text{otherwise,} \end{cases}$$

where we have set $x_0^e = \mathbf{v}_a^e$ and $x_{n_e}^e = \mathbf{v}_b^e$. We can define the neighboring set $\mathbf{W}_v$ of a vertex $\mathbf{v} \in \mathbf{V}$ as the union of all the sets $[\mathbf{v}_a^e, x_1^e]$ and $[x_{n_e}^e, \mathbf{v}_b^e]$, $\mathbf{e} \in \mathbf{E}_v$. This way $\mathbf{W}_v$ will be a star-shaped metric graph itself.

We can introduce the functions $\phi_v(x)$ with support $\text{supp}(\phi_v(x)) = \mathbf{W}_v$ such that

$$\phi(x)|_{\mathbf{W}_v \cap \mathbf{e}} = \begin{cases} 1 - \frac{|x_v^e - x^e|}{h_e} & \text{if } x^e \in \mathbf{W}_v \cap \mathbf{e}, \\ 0 & \text{otherwise,} \end{cases} \quad \mathbf{e} \in \mathbf{E}_v,$$

where x_v^e is either 0 or ℓ_e depending on the direction on the edge and its parametrization.

Using these elements, shown on Figure 1, the authors of [4] define a finite-dimensional discrete system of equations, which has a unique solution and proceed to prove several properties on the discretized Hamiltonian $\mathbf{H}$ corresponding to the equation system.

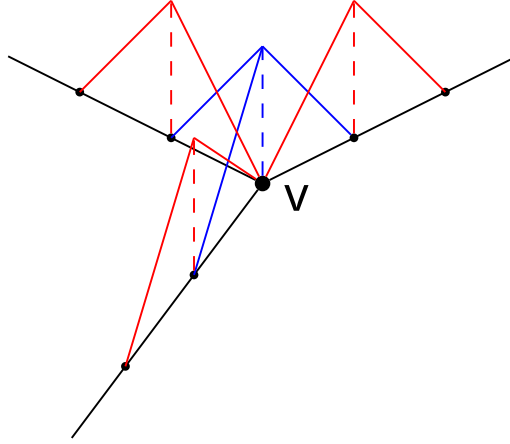


Figure 1. Blue: special finite element for vertices. Red: classical finite elements on the edges.

First, the following further notations are introduced:

$$\mathbf{u}_E = \begin{bmatrix} \mathbf{u}^1 \\ \vdots \\ \mathbf{u}^M \end{bmatrix}, \quad \text{where} \quad \mathbf{u}^e = \begin{bmatrix} u_1^e \\ \vdots \\ u_{n_e-1}^e \end{bmatrix} \quad \mathbf{e} = (1, \dots, M), \quad \text{and} \quad \mathbf{u}_V = \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}.$$

Let $\mathbf{f}_E$ and $\mathbf{f}_V$ be the vectors

$$\mathbf{f}_E = \begin{bmatrix} \mathbf{f}^1 \\ \vdots \\ \mathbf{f}^M \end{bmatrix}, \quad \text{where} \quad \mathbf{f}^e = \begin{bmatrix} f_1^e \\ \vdots \\ f_{n_e-1}^e \end{bmatrix}, \quad \text{and} \quad \mathbf{f}_V = \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix},$$

with $f_k^e = \int_{\text{supp}(\psi_k^e)} f \psi_k^e dx$ and $f_v = \int_{W_v} f \phi_v dx$. Then the mentioned system of equations can be expressed as

$$\mathbf{H} \begin{bmatrix} \mathbf{u}_E \\ \mathbf{u}_V \end{bmatrix} = \begin{bmatrix} \mathbf{f}_E \\ \mathbf{f}_V \end{bmatrix},$$

where $\mathbf{H}$ has the following structure:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21}^\top & \mathbf{H}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{G} \end{bmatrix} + \begin{bmatrix} \mathbf{V} & \mathbf{C} \\ \mathbf{C}^\top & \mathbf{F} \end{bmatrix} = \mathbf{L} + \mathbf{M}_v.$$

In [4], the authors prove the following properties:

1. $\mathbf{H}$ is a symmetric positive definite matrix.
2. $\mathbf{A}$ and $\mathbf{V}$ are block diagonal matrices and each diagonal block can be permuted into a symmetric tridiagonal nonsingular matrix corresponding to the edge e .
3. The entries of $\mathbf{A}$ are given by

$$\int_e \frac{d}{dx} \psi_j^e \frac{d}{dx} \psi_k^e dx = \begin{cases} 2/h_e & \text{if } j = k, \\ -1/h_e & \text{if } |j - k| = 1, \\ 0 & \text{otherwise,} \end{cases}$$

and the entries of $\mathbf{B}$ are given by

$$\int_{W_v} \frac{d}{dx} \phi_v \frac{d}{dx} \psi_k^e dx = \begin{cases} -1/h_e & \text{if } e \in W_v \cap E_v \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, $\mathbf{G}$ is diagonal and the entries of $\mathbf{G}$ are given by

$$\int_{W_v} \frac{d}{dx} \phi_v \frac{d}{dx} \phi_v dx = \sum_{e \in E_v} 1/h_e.$$

4. The entries of $\mathbf{V}$ are $\int_e \psi_j^e \psi_k^e v(x) dx$, the entries of $\mathbf{C}$ are $\int_{W_v} \phi_v \psi_k^e v(x) dx$, the entries of $\mathbf{F}$ are $\int_{W_v} \phi_v \phi_v v(x) dx$ and $\mathbf{F}$ is diagonal.

Furthermore, for a more detailed derivation and highlighting specific results, we refer to [4]. It elaborates on the conventional quadratic convergence rate, where the error is proportional to $\mathcal{O}(h^2)$, as well as the linear convergence of the Neumann-Kirchhoff condition, where the error is proportional to $\mathcal{O}(h)$, with h being the maximum of h_e across all edges.

This framework provides a tool for numerically solving the proposed problem on a quantum graph which is priorly known. However, complications may arise once we change the structure of the graph. FEM uses sparse matrices to store the graph, so altering the structure is computationally expensive. Additionally, if the problem has high space complexity, the graph may not fit into memory. Another issue arises if the graph's structure is unknown beforehand. For these reasons, we search for further solutions.

2.3 Neumann-Neumann method for quantum graphs

Domain decomposition methods (DDMs) divide large-scale computational problems into smaller subdomains that can be solved independently or in parallel, enhancing scalability and speeding up the solution process.

In this section we describe the Neumann-Neumann method for quantum graphs based on [25]. A given G quantum graph is decomposed into disjoint (w.r.t. their edges) subgraphs $\{G_i = (V_i, E_i)\}_{i=1,2,\dots,L}$.

The idea of Neumann-Neumann methods is to keep track of the interface values and iteratively update these values based on the deviation from the Neumann-Kirchhoff condition. Formally, we start the algorithm from a zero (or any inexpensive) initial guess u_G^0 . For $n \geq 0$ the new iterate is computed as follows: first we solve the Dirichlet problems

$$\begin{aligned} f_e(x) &= -(c_e u_e^{n+\frac{1}{2}})'(x) + v_e(x) u_e^{n+\frac{1}{2}}(x), \quad x \in (0, \ell_e), \quad e \in E_i, \\ 0 &= I_v U_i^{n+\frac{1}{2}}(v), \quad v \in V_i \setminus G, \\ u_G^n(v) &= U_i^{n+\frac{1}{2}}(v), \quad v \in V_i \cap G, \\ 0 &= C_i(v)^\top U_i^{n+\frac{1}{2}}(v), \quad v \in V_i \setminus G. \end{aligned} \tag{1}$$

Here the function C_i is the restriction of C to G_i . Note that we impose natural boundary conditions on the set of vertices $\partial G_i \cap \partial G$, but we will still refer to these problems as Dirichlet problems. Then we compute the solutions of the residual Neumann problems

$$\begin{aligned} 0 &= -(c_e w_e^{n+1})'(x) + v_e(x) w_e^{n+1}(x), \quad x \in (0, \ell_e), \quad e \in E_i, \\ 0 &= I_v W_i^{n+\frac{1}{2}}(v), \quad v \in V_i \setminus G, \\ 0 &= C_i(v)^\top W_i^{n+1'}(v), \quad v \in V_i \setminus G, \\ \sum_{i:v \in V_i} C_i(v)^\top U_i^{n+\frac{1}{2}}(v) &= C_i(v)^\top W_i^{n+1'}(v), \quad v \in V_i \cap G. \end{aligned} \tag{2}$$

Finally, we update the interface values as

$$u_G^{n+1}(v) = u_G^n(v) - \theta \sum_{e \in E_v} w_e^{n+1}(v), \quad v \in G, \tag{3}$$

with an appropriate $\theta \in (0, \theta_{\max})$, for some $\theta_{\max} > 0$ [43, Chapter C.3].

In [25] it is shown that the discrete finite element version of the above procedure can be rewritten as a preconditioned Richardson iteration for the Schur complement of the finite elements corresponding to the vertices. We note that the usual presentation of domain decomposition methods is based on the Richardson iteration mainly for the sake of readability, but in practice, the Schur complement system is often solved with more sophisticated algorithms, such as GMRES [35] or BiCGSTAB [44].

GMRES is an iterative method designed for solving non-symmetric linear systems. It minimizes the residual norm at each step over a Krylov subspace, which in linear algebra is defined as follows: the order- r Krylov subspace generated by an $n \times n$ matrix $\mathbf{A}$ and a vector $\mathbf{b}$ of dimension n is the linear subspace spanned by the images of $\mathbf{b}$ under the first r powers of $\mathbf{A}$ (starting from $\mathbf{A}^0 = \mathbf{I}$), that is,

$$\mathcal{K}_r(\mathbf{A}, \mathbf{b}) = \text{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \dots, \mathbf{A}^{r-1}\mathbf{b}\}.$$

This subspace captures the action of $\mathbf{A}$ on $\mathbf{b}$, allowing GMRES to find approximate solutions efficiently without requiring explicit matrix representations. However, GMRES can be memory-intensive as it stores all previous iterates, leading to high computational costs in large-scale problems. Restart strategies are often implemented to balance efficiency and memory usage.

The GMRES algorithm is derived from the Arnoldi process, which generates an orthonormal basis for these Krylov subspaces. This process results in an upper Hessenberg matrix representation that facilitates solving linear systems efficiently. GMRES has several advantages over

other methods, such as Generalized Conjugate Residual (GCR) and ORTHODIR, particularly its robustness against breakdown for indefinite problems unless it has already converged. Furthermore, GMRES requires less storage than GCR and fewer arithmetic operations, making it suitable for large-scale applications.

BiCGSTAB, on the other hand, is specifically designed for large nonsymmetric systems and combines features from both the Conjugate Gradient and GMRES methods. It stabilizes convergence by utilizing two sequences of residuals and iterates, which mitigates oscillatory behavior often encountered in earlier methods. By operating within Krylov subspaces, BiCGSTAB effectively generates iterates that enhance convergence stability while maintaining lower memory requirements compared to GMRES.

Both GMRES and BiCGSTAB integrate effectively within the Neumann-Neumann method framework to enhance computational efficiency and stability in solving PDEs on complex domains such as quantum graphs. These iterative solvers are compatible with matrix-free approaches, avoiding explicit formation or storage of the system matrix. This is particularly advantageous for large-scale problems, reducing memory requirements by computing matrix-vector products on-the-fly, which facilitates efficient handling of complex geometries and boundary conditions in PDEs. Additionally, leveraging neural networks as matrix-free solvers introduces further computational efficiency, as they enable solution approximation without assembling a global matrix, offering greater flexibility and scalability. Both GMRES and BiCGSTAB can thus be employed in matrix-free contexts, streamlining computation for large or intricate systems.

2.4 Physics-Informed Neural Networks

When employing PINNs, practitioners typically operate with small datasets and employ simple model architectures, often without additional regularization measures such as dropout, yet consistently achieve excellent results. We illustrate the application of PINNs with an example from the foundational paper that introduced this framework [34].

Applying this paradigm manifests in the loss function of the training algorithm. Consider a partial differential equation of the general form

$$\frac{\partial u}{\partial t} + \mathcal{N}[u] = 0, \quad x \in \Omega, \quad t \in [0, T],$$

where $u(t, x)$ represents the solution and $\mathcal{N}$ denotes a possibly nonlinear differential operator, with $\Omega \subset \mathbb{R}^D$. We define $f(t, x)$ as

$$f := \frac{\partial u}{\partial t} + \mathcal{N}[u],$$

and approximate $u(t, x)$ using our neural network. Utilizing the chain rule for differentiating compositions of functions through automatic differentiation, we can compute $f(t, x)$ and incorporate this information into the loss function of our network. Therefore, while the conventional loss function typically comprises a single mean square error (MSE) loss based on the prediction of the solution against the ground truth, we now include

$$\text{MSE} = \text{MSE}_u + \text{MSE}_f,$$

where

$$\text{MSE}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

and

$$\text{MSE}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

Here, $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary data, and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ represent the collocation points for $f(t, x)$. Consequently, MSE_u corresponds to the training data, while MSE_f enforces the physical constraints inherent in the problem.

This approach provides a robust framework for solving complex physical problems with limited data while seamlessly integrating domain knowledge. By leveraging both data-driven insights and fundamental physical principles, PINNs offer a promising avenue for addressing various scientific challenges.

3 Implementation

3.1 Overview

To overcome the main difficulty of existing PINN-based methods for quantum graphs, the explicit dependence on the graph structure, we use the Neumann-Neumann iteration introduced in Section 2.3. For the sake of versatility, we completely decompose the graph to its edges. Since the Neumann-Neumann iteration consists of Dirichlet problems and Neumann problems, we need to train a PINN for a one-dimensional problem with Dirichlet boundary condition on both sides and a PINN for a one-dimensional problem with Neumann boundary condition on both sides. However, the original problem might contain classical one-dimensional Neumann boundary conditions imposed on vertices of $\partial\mathbf{G}$, and thus we need to train two more PINNs corresponding to edges with Dirichlet-Neumann and Neumann-Dirichlet boundary condition configurations.

We note that we could introduce further elementary subgraphs to the decomposition, for example, we could decompose the graph into triangles and edges. This might improve convergence, but the number of separate PINNs that need to be trained also increases.

3.2 Training data generation

In the implementation phase, the process begins with the generation of training data. For this mathematical problem, we use synthetically generated data, as real-world measurements are not necessary to achieve generalization for the neural network.

The training data consists of inputs and outputs, where the inputs are the right-hand side of the differential equation (f_e), the numerical values for the two boundary conditions (BC values), and the two coefficients appearing in the equation (c_e and v_e). Once the inputs are ready, we can use a 1D FEM solver to generate the outputs, which will be the reference solution for the neural network.

3.2.1 Right-hand side

For the right-hand side of the PDE, we employ a Matérn kernel, a function associated with Gaussian random fields (GRFs), to model the relationships between data points. GRFs serve as the foundation for this approach, as they define distributions over functions and allow for the generation of training data. A GRF is essentially a collection of random variables, where each point in a space is associated with a normally distributed random variable. It is fully specified by its mean and covariance functions, with the Matérn kernel serving as a popular choice for specifying the covariance structure.

The Matérn kernel is characterized by two key parameters: smoothness (ν) and length scale (l). Smoothness controls the rate of change in the generated functions, with higher values corresponding to smoother functions, while the length scale determines the spatial extent of

correlations between data points. In our implementation, we set $\nu = 2.5$ (corresponding to twice differentiable functions) and $l = 0.1$.

The formula of the Matérn kernel is given by:

$$k(x_i, x_j) = \frac{1}{\mathbf{G}(\nu)2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{L} |x_i - x_j| \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}}{L} |x_i - x_j| \right),$$

where $K_\nu(\cdot)$ is a modified Bessel function, and $\mathbf{G}(\cdot)$ is the Gamma function.

GRFs provide a structured approach to uncertainty quantification, as any finite subset of values drawn from a GRF follows a multivariate Gaussian distribution. By sampling from GRFs with the Matérn kernel, we generate independent and identically distributed samples that capture the underlying distribution required for training our neural networks. This approach ensures that models are trained on a diverse and representative dataset of the system's dynamics, enhancing their predictive accuracy and capacity for rigorous uncertainty quantification.

We use a fixed discretization of $N = 128$ points and aim to obtain samples within the spatial interval $[0, \ell]$, where we set $\ell = 1$. This choice of $\ell = 1$ ensures uniformity and simplifies the computational process without loss of generality, as we transform the graph into an equilateral configuration. Selecting $N = 128$ provides sufficient resolution for accurately capturing the behaviour of the system while maintaining computational efficiency. Subsequently, we generate multiple samples from a multivariate normal distribution characterized by a zero-mean vector and a computed covariance matrix. These samples effectively represent the right-hand side of the PDEs.

The image in Figure 2 illustrates the distribution of samples obtained from the right-hand side values.

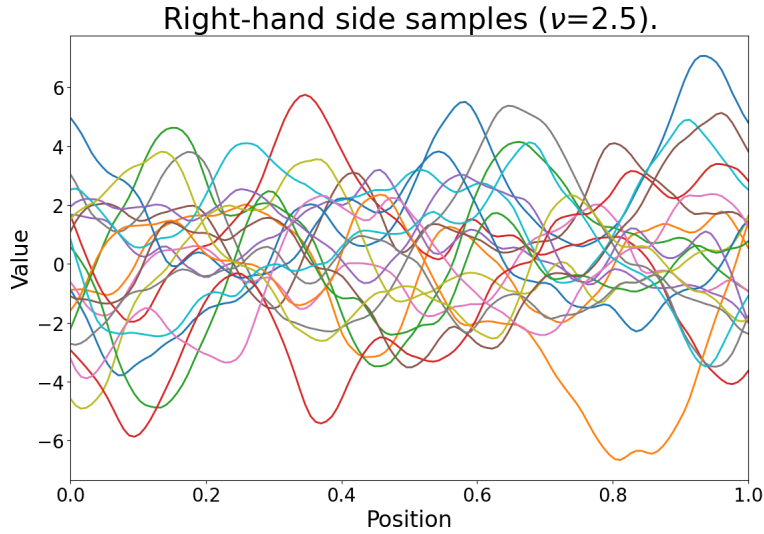


Figure 2. 20 samples from the generated right-hand sides

3.2.2 Boundary condition values

Each input requires two boundary conditions, which can be of Dirichlet or Neumann type. Correspondingly, we develop four distinct neural network models to handle each possible combination of these boundary conditions, selecting the appropriate model for each edge during evaluation. When generating the training data, our primary concern is the numerical value of the boundary conditions. We generate these values as random numbers uniformly distributed within the interval $[-10, 10]$. This range ensures sufficient variability in the boundary conditions.

3.2.3 Coefficients c and v

In our implementation, the coefficients c and v are treated as function inputs, allowing them to be provided as functions discretized into $N = 128$ points, similar to how the function f is handled. While we initially simplify the treatment of these coefficients by assuming c is set to a constant value of 1 and v is chosen uniformly at random from the range $[0.1, 5]$, they are ultimately implemented as input vectors for future extensibility in model training.

Although these coefficients can vary spatially in real-world applications, we adopt this simplification to facilitate initial proof-of-concept development. This approach enables us to concentrate on developing the core functionality of the system, ensuring stability and feasibility during experimentation. However, it is crucial to recognize that this simplification imposes limitations on the model's ability to capture spatial variations in these coefficients, which are often significant in practical scenarios.

3.2.4 Reference solution

Before applying the FEM solver, inputs must be preprocessed to account for varying quantities of right-hand side and boundary condition values, enabling flexible dataset generation. This flexibility is achieved by taking the Cartesian product of the right-hand side and boundary condition sets. The v coefficient values are not included as a separate set to avoid creating an excessively large dataset; instead, these v values are repeated as needed to match the size of the dataset generated from the Cartesian product.

Following the creation of all the inputs, the next step is to generate solutions for the associated PDEs using 1D FEM. The visualized solutions can be seen in Figure 3.

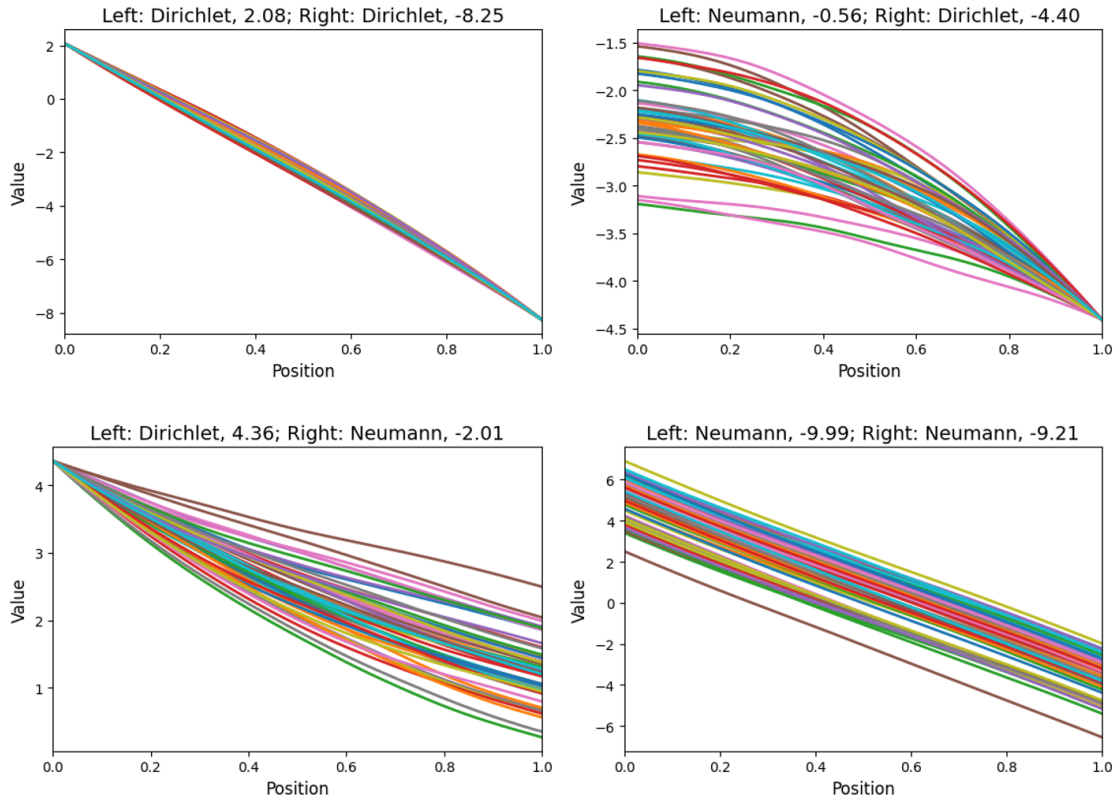


Figure 3. Visualization of 50 solution samples per edge type, illustrating the impact of shared boundary conditions within each subset.

Upon analyzing the solutions, it becomes visible how boundary condition types and values significantly impact the system compared to the right-hand sides.

The generated dataset is partitioned into training and validation datasets for neural network training. However, it is imperative to ensure data segregation between these sets, preventing any overlap or duplication of data instances between them. This segregation is achieved by using distinct randomization seeds for the selection of samples in each set, ensuring that the training and validation datasets remain independent.

3.3 Model

In the PINN framework, four models were trained, each tailored to a specific edge type resulting from distinct boundary condition combinations (DD, DN, ND, NN).

To maintain simplicity and serve as a proof-of-concept, a basic fully connected structure with tanh activation functions was explored for the network architecture (Figure 4). The input layer of the network has dimension $3 \cdot 128 + 3 = 387$, consisting of the right-hand side f_e of size 128, coefficients c_e and v_e each of size 128, boundary data of size 2 and spatial coordinate x of size 1. The output is the solution at the point x . We note that including spatial information is essential to compute the network’s gradient w.r.t. to x so that we can compute the derivative for the loss function. The network architecture is shown on Figure 4.

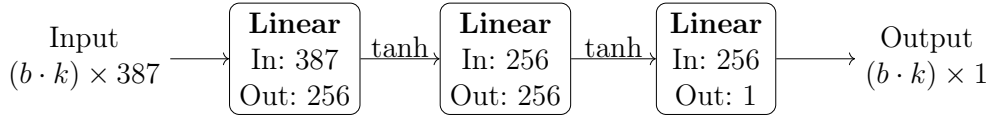


Figure 4. Network architecture. b is the batch size, k is the number of x coordinates given as input.

Despite initial attempts to incorporate a dot product layer inspired by the work of Wang et al. [45] this approach did not yield the desired results in our experiments. Consequently, the final network architecture adopted a simpler design as mentioned.

Including spatial information allows for the calculation of gradients with respect to spatial variables, essential for incorporating the physics-informed part of the loss function. By calculating gradients with respect to the spatial variable, the network can substitute back into the PDE, ensuring that the solutions generated adhere closely to the governing equations. This integration of physics-based constraints into the training process improves the network’s ability to generalize and produce accurate predictions across a range of conditions.

3.4 Training

3.4.1 Training setup

For effective training and experimentation, ensuring reproducibility and determinism is critical. To achieve this, we set seeds for all components of the code, rendering the process deterministic and repeatable. Additionally, we leverage the capabilities of the *Weights & Biases* (W&B) [7] library, which assists in tracking and visualizing results, versioning artifacts, and conducting hyperparameter sweeps.

To ensure model persistence and enable recovery from interruptions, model checkpointing is employed. Periodically, after a fixed number of epochs, the current state of the model, including its parameters and optimizer state, is saved to disk. This allows training to resume from the last saved checkpoint, preserving the progress made during training. Additionally, the best-performing model based on validation loss will be saved separately for later evaluation and deployment.

For the optimization algorithm, we employ the AdamW optimizer [30], known for its efficacy in training deep neural networks and its handling of large-scale optimization problems. AdamW extends the Adam optimizer [22] by integrating weight decay directly into the optimization step. This approach maintains adaptive learning rates for each parameter and incorporates L2 regularization in the optimization process, mitigating overfitting by penalizing large weights.

Hyperparameter tuning is crucial for model convergence and generalization. These parameters, including the initial learning rate and weight decay, are fine-tuned through experimentation and validation on a separate dataset. In our implementation, we set the initial learning rate to $5 \cdot 10^{-4}$ and weight decay to $1 \cdot 10^{-5}$, based on their effectiveness in our problem domain. However, optimal settings may vary depending on dataset characteristics and model architecture.

Learning rate scheduling plays an essential role in optimizing the training process by adaptively adjusting the learning rate during training. A common strategy is to reduce the learning rate when the validation loss plateaus, allowing the optimizer to converge more effectively towards the minima of the loss function. The learning rate scheduler monitors the validation loss and updates the learning rate accordingly. In our implementation, we utilize a reduction factor of 0.7 after every 15 epoch of patience, updating it only if the validation loss fails to decrease by a threshold of 0.0001. The choice of these specific values, including the reduction factor, patience, and threshold, was inspired by [45], which demonstrated their effectiveness in facilitating efficient training. This meticulous adjustment of the learning rate aids in preventing overfitting and ensures smooth convergence towards the optimal solution. Additionally, the training process is terminated if the learning rate falls below $1 \cdot 10^{-7}$, serving as an early stopping criterion to prevent unnecessary computations.

3.4.2 Data loading and preprocessing

The training data consists of input-output pairs representing the right-hand side of the partial differential equation (PDE), boundary conditions (BC), and physical parameters c and v (where c is currently always set to constant 1 and v is constant but varying random number from $[0.1, 5]$). Upon loading the dataset, it undergoes preprocessing steps to be compatible with the model. Additionally, the ratio between the training and validation set, set at 80% training and 20% validation, is carefully maintained to ensure adequate model evaluation and generalization performance.

The training data consists of input-output pairs representing the right-hand side of the partial differential equation (PDE), boundary conditions (BC), and physical parameters c and v , where c is a constant set to 1, and v is a constant but varies randomly within the range $[0.1, 5]$. Upon loading, the dataset undergoes preprocessing to ensure compatibility with the model. An 80%-20% split is maintained between the training and validation sets to support robust model evaluation and generalization.

Mini-batch training is used, meaning the dataset is divided into small batches, and gradients are computed and applied to update the model parameters iteratively. The selection of batch size involves empirical considerations informed by both hardware constraints and dataset attributes. While a batch size of $b = 2048$ was predominantly employed, variations such as 512 and 1024 were also used to accommodate the available GPU memory on the cluster.

As illustrated in Figure 5, the preprocessing of the dataset involves several transformations to prepare the data for neural network training. When generating the data, the following components are obtained:

The right-hand sides (f) are generated as matrices of size $M \times N$, where M denotes the size of the dataset and N represents the discretization dimension of f . Boundary condition values (BC) are stored as matrices of size $M \times 2$. The constants (c and v) are also stored as $M \times N$ matrices. Additionally, the reference output (y) is generated as matrices of size $M \times N$.

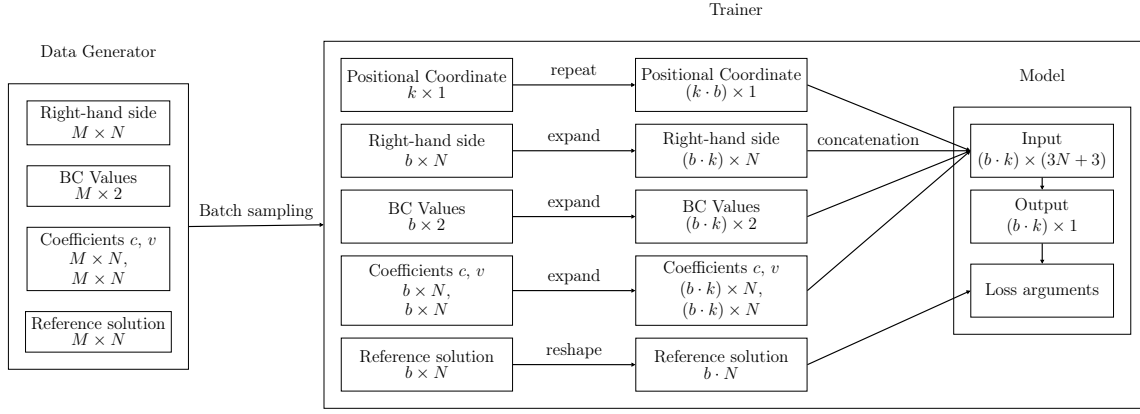


Figure 5. Diagram illustrating the components of the code, showing the flow of data and the changes in dimensions. The functions are discretized into N points, M represents the overall size of the dataset, b is the batch size, while k is the number of positional coordinates given.

All these components are saved in files, preserving the specified dimensions for the training dataset.

During training, a batch of data with a batch size of b is loaded, along with the positional coordinate x , which is necessary for inputs. The coordinate x is discretized into N points representing the interval $[0, 1]$, which is essential for calculating the derivative of the loss with respect to x . Thus, x is generated as a one-dimensional array of size $N \times 1$.

It is important to note that the output of the model corresponds to the predictions at the specified spatial coordinates x , which has a dimensionality of k . During training, we use $k = N$ spatial points. However, after training, the model can generate predictions at any number of coordinates, and for any value on the interval $[0, 1]$. For the Neumann-Neumann iteration, only the interface information is required, which can be obtained by evaluating the model at 4 points (2 per side, to capture the necessary Neumann information). Although reducing the number of input parameters does not significantly reduce inference time, it allows us to explore alternative approaches. One such approach involves aggregating all the data into a single input array, enabling one full iteration of the Neumann-Neumann method to be computed with a single pass through each model. This could potentially improve efficiency, although we did not explore this option yet.

During batch processing, NumPy's `repeat`, `expand`, and `reshape` methods are employed to transform the data into a format compatible with the model, as illustrated in Figure 5. The coordinate x is transformed from a $k \times 1$ array into an $(k \cdot b) \times 1$ array by repeating the coordinates for each batch. The right-hand sides (f) are expanded from a $b \times N$ array to an $(b \cdot k) \times N$ array to match the batch size, where N is the discretization dimension for f . Boundary condition values (BC) are reshaped from a $b \times 2$ array to an $(b \cdot k) \times 2$ array to align with the positional coordinates. The coefficients are similarly transformed into arrays of shape $(b \cdot k) \times N$. Finally, the reference output (y) is transformed from a $b \times N$ array to an $b \cdot N$ array. This transformation process ensures that all data components are aligned appropriately for the model.

3.4.3 Loss function

The choice of an appropriate loss function plays a pivotal role in training PINN models. In this work, we employ a combination of mean squared error (MSE) loss components aimed at optimizing the model parameters while ensuring adherence to the underlying physical laws governing the system.

The primary component of our loss function is the mean squared error (MSE) loss, a widely-used metric for regression tasks. Mathematically, the MSE loss is defined as the average of the squared differences between predicted and actual values:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

where N is the number of data points, y_i represents the ground truth value, and $\hat{y}_i$ denotes the predicted value. By minimizing the MSE loss, we aim to minimize the discrepancy between the predicted and actual solutions, thereby improving the overall accuracy of the model.

In addition to the MSE loss measuring the discrepancy between predictions and ground truth, our loss function incorporates MSE loss terms aimed at enforcing adherence to the underlying physical constraints of the problem. These components penalize deviations from the governing equation and boundary conditions, ensuring that the model produces solutions that adhere to the laws of physics.

The used loss function for the PINN with Dirichlet-Neumann boundary is as follows:

$$\begin{aligned} L^{DN}(\theta) &= \alpha \cdot L_1(\theta) + \beta \cdot L_2(\theta) + \gamma \cdot L_3^{DN}(\theta) + \delta \cdot L_4(\theta), \\ L_1(\theta) &= \frac{1}{N} \sum_{i=1}^N (u(x_i) - \mathcal{N}(x_i|\theta))^2, \\ L_2(\theta) &= \frac{1}{N} \sum_{i=1}^N (-c\mathcal{N}''(x_i|\theta) + v\mathcal{N}(x_i|\theta) - f(x_i))^2, \\ L_3^{DN}(\theta) &= (u(x_0) - \mathcal{N}(x_0|\theta))^2 + (u'(x_N) - \mathcal{N}'(x_N|\theta))^2, \\ L_4(\theta) &= |\theta|. \end{aligned}$$

Here N is the number of data points, u denotes the ground truth (generated by FEM) and $\mathcal{N}$ denotes the output of the network.

The individual loss terms can be explained as follows:

- $L_1(\theta)$ is the network's error predicting the available data,
- $L_2(\theta)$ is the residual error enforcing $\mathcal{N}$ to satisfy the underlying PDE,
- $L_3(\theta)$ is the boundary error,
- $L_4(\theta)$ is a Tikhonov regularization to prevent overfitting,
- α, β, γ and δ are constants that control the contribution of each term.

For the rest of the PINNs the L_1 , L_2 and L_4 terms are unchanged, while the boundary error is computed as

$$\begin{aligned} L_3^{DD}(\theta) &= (u(x_0) - \mathcal{N}(x_0|\theta))^2 + (u(x_N) - \mathcal{N}(x_N|\theta))^2, \\ L_3^{NN}(\theta) &= (u'(x_0) - \mathcal{N}'(x_0|\theta))^2 + (u'(x_N) - \mathcal{N}'(x_N|\theta))^2, \\ L_3^{ND}(\theta) &= (u'(x_0) - \mathcal{N}'(x_0|\theta))^2 + (u(x_N) - \mathcal{N}(x_N|\theta))^2. \end{aligned}$$

The weights assigned to each component are determined empirically and through hyperparameter tuning, reflecting their relative importance in achieving the desired model performance. In our implementation, we set the values of $\alpha = 1$, $\beta = 1 \cdot 10^{-3}$, $\gamma = 1$ and $\delta = 0$. We note, that we included the boundary error explicitly, since the numerical stability of the Neumann-Neumann method heavily depends on the precision at the boundaries.

3.4.4 Hyperparameter tuning

Fine-tuning hyperparameters is essential in training neural networks, including PINNs, due to their complexity and sensitivity. Our approach to hyperparameter tuning centered on heuristic experimentation.

This heuristic approach involved manually adjusting hyperparameters based on the observed performance of the model. Through iterative refinement, we developed an intuitive understanding of how each hyperparameter influenced the training dynamics and eventual outcomes of our PINNs.

While our heuristic method yielded promising results, we acknowledge the potential for further improvement through more advanced hyperparameter tuning techniques. Methods such as grid search and Bayesian optimization offer systematic and automated approaches for exploring the hyperparameter space.

Implementing these advanced methodologies in future experiments could reveal superior hyperparameter configurations, thereby enhancing the performance and efficiency of our PINNs.

3.4.5 Computational considerations

The training process was executed on a high-performance computing cluster featuring Nvidia A100 and V100 GPUs. The duration of each training epoch varied depending on the model's complexity and the dataset size. For the final training runs with the optimized models and datasets, each epoch took approximately 10-15 minutes.

One of the significant challenges encountered during training was the extensive duration, especially when dealing with larger datasets. This prolonged training time required the implementation of robust checkpointing mechanisms to ensure the ability to resume training seamlessly in the event of interruptions. By periodically saving model checkpoints, we could safeguard against potential disruptions, allowing us to continue training from the last saved checkpoint without losing progress.

Moreover, computational performance metrics were closely monitored to assess resource utilization and enhance training efficiency. Key metrics such as memory usage and epoch training time were carefully tracked throughout the training process. These metrics offered valuable insights into resource requirements and facilitated the optimization of training strategies for improved performance.

4 Numerical experiments and results

In this section, we discuss our extensive experimentation with various architectures, training dataset sizes, and other hyperparameters. Our goal was to understand the performance and limitations of the PINN models in solving PDEs under different conditions.

4.1 Training data

Initially, we discretized our functions into $N = 1024$ points, but found that lowering this to $N = 128$ drastically reduces training times without sacrificing accuracy.

At first, we trained and tested our models with both c_e and v_e set to a constant value of 1. Later we also trained a set of PINNs where v_e was a constant function with different values, although these models exhibited lower accuracy, especially when encountering values outside the range on which they were trained.

We conducted thorough testing on the number of boundary conditions and right-hand side samples required for training. Initially, we tested the models using a single pair of boundary

conditions (BCs) and multiple right-hand side samples, which were successful. Subsequently, we evaluated the models’ generalization capability with a fixed right-hand side and various boundary condition pairs, but this proved unsuccessful. Our investigation also involved training outcomes with mixed datasets, initially yielding unsuccessful results. However, we later determined that a total of $3 \cdot 10^6$ samples (derived from the Cartesian product of 1,000 boundary condition values and 3,000 right-hand sides) was sufficient to achieve accuracy comparable to FEM.

However, we wish to highlight that our findings show that the performance of the pretrained PINNs did not plateau and increasing the number of training samples could further increase its accuracy.

4.2 Network architecture and loss function

Our experiments indicate that increasing the number and width of the linear layers improves model accuracy. However, these accuracy gains come with longer training times, demonstrating the typical trade-off in neural network design. To achieve a practical balance between accuracy and training efficiency, we limited the layer count and width to avoid excessively long training durations.

We tested some other architectures, for example, the so-called linearity-preserving network of [45], which has a similar structure to that of the branch and trunk network of DeepONet. However, we did not see enough improvements to switch from the basic linear layer setup.

Additionally, we explored different α , β , G and δ coefficients for the various loss types; however, we found the current configuration to be the most stable, achieving convergence across a wide range of datasets without overfitting or underperforming.

4.3 Richardson iteration

We found that the convergence rate of the Richardson iteration is extremely sensitive to the θ parameter, even if the edges are solved with traditional FEM. Most notably, the increased iteration number in the case of the Barabási graph is due to the fact that setting anything above the very low value of $\theta = 0.066$, the iteration became unstable. In the future, we wish to switch to a more sophisticated algorithms, as the mentioned GMRES or BiCGSTAB to avoid the tedium of finding an optimal θ value.

4.4 Results of 1D assessment

As illustrated in Table 1 and depicted in Figure 6, with the dataset and hyperparameters specified, our models achieve error rates comparable to FEM solutions. However, this improvement comes at the cost of significantly increased training times. It is also important to note that the statistic tracking library used (Weights&Biases) does not accurately record training time if interruptions occur. Given the numerous interruptions experienced during our training, the reported training times may contain inconsistencies.

Table 2 shows that allowing v_e to be constant but vary across edges leads to an increase in error by approximately one order of magnitude. This suggests a more complex problem than the previous case, potentially requiring additional data or layers to achieve similar accuracy. However, both options would extend training times significantly, which are already higher than in the previous scenario.

These results demonstrate that with a well-structured dataset and carefully chosen hyperparameters, our neural networks can effectively learn the target task, thus facilitating further experiments on quantum graphs.

Type	Training time	Ground truth MSE	Best val. loss	Nr. BC	Nr. f	Epoch
NN	11d 4h 22m 45s	1.716e−6	4.098e−5	1000	3000	2000
ND	14d 15h 2s	9.004e−7	1.872e−5	1000	3000	2000
DN	16d 5h 30m 56s	9.081e−7	2.259e−5	1000	3000	2000
DD	10d 21h 44m 27s	5.477e−7	1.110e−5	1000	3000	2000

Table 1. Performance metrics for training with dataset where $v_e(x) \equiv 1$.

Type	Training time	Ground truth MSE	Best val. loss	Nr. BC	Nr. f	Epoch
NN	18d 2h 39m 9s	7.1101−5	9.1064−4	1000	3000	2286
ND	17d 19h 16m 9s	6.0751−6	1.1909−4	1000	3000	1272
DN	17d 19h 20m 58s	3.6147−6	6.6651−5	1000	3000	2113
DD	17d 13h 50m 38s	2.1116−6	9.1064−4	1000	3000	2070

Table 2. Performance metrics for training with dataset where v_e is constant but varying. There were 3000 values for v_e in each case.

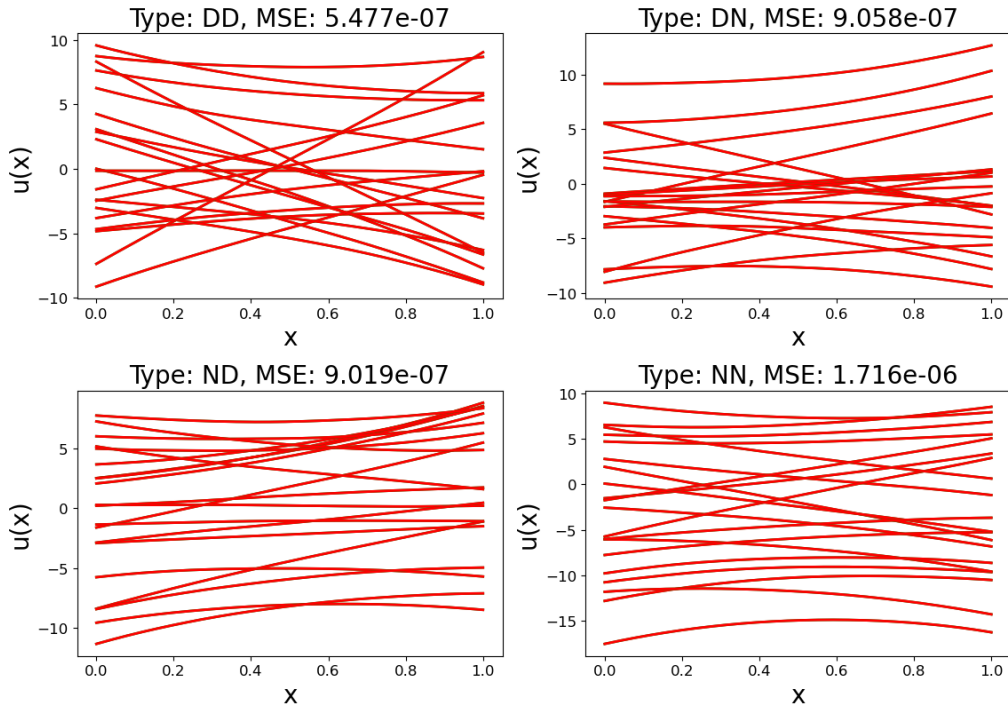


Figure 6. Comparison of ground truth and predicted solutions: This image illustrates a visual comparison between the ground truth (depicted in green) and the corresponding predictions (depicted in red) for 20 samples from the validation set. Each data point represents a solution obtained from the model, showcasing the agreement or deviation between the predicted and actual values. Visualized in the $v_e(x) \equiv 1$ case.

4.5 Results on graphs

We tested our algorithm on the following graph structures:

- A *star* graph, consisting of a center node connected to $n = 3$ outer nodes, is a fundamental structure in the literature of quantum graphs.
- The *ladder* graph consists of two paths of $n = 100$ nodes, with each pair connected by a single edge. Thus, it has $2n$ nodes and $3n - 2$ edges.
- The *Turan* graph is a complete multipartite graph on $n = 13$ nodes with $r = 4$ disjoint subsets. That is, edges connect each node to every node not in its subset.
- The *Paley* $\frac{(p-1)}{2}$ -regular graph on $p = 23$ nodes was also used. Generally, Paley graphs are constructed from the members of a suitable finite field by connecting pairs of elements that differ by a quadratic residue.
- While the performance of the algorithm on well-structured graphs like the ones above is important, we wanted to test its robustness on larger, random generated graphs. We chose the *Barabási* model, which generates a random graph generated through the Barabási-Albert preferential attachment model, in which a graph of $n = 100$ nodes is grown by attaching new nodes each with $m = 2$ edges that are preferentially attached to existing nodes with high degree. The degree distribution of the resulting graph follows a power law.

Figure 7 shows the structure of the used graphs.

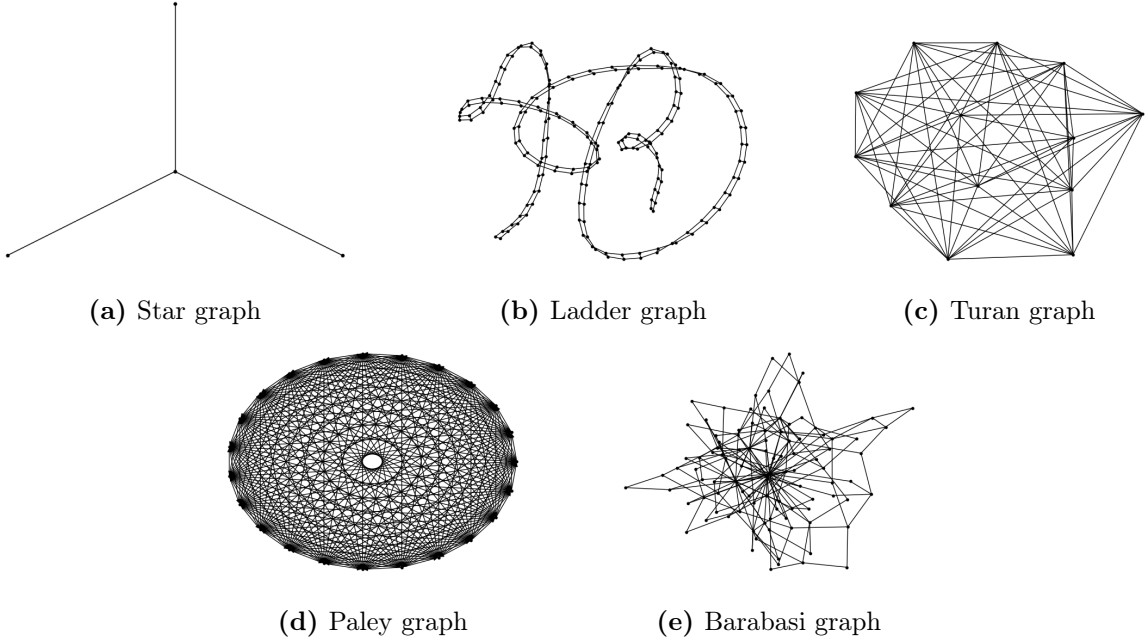


Figure 7. Topology of graphs used in the numerical experiments

The graphs above were generated using the ‘networkx’ package. To evaluate the numerical accuracy, we calculated various error metrics including the L^2 error, edgewise mean L^2 error, maximum error, and Neumann-Kirchhoff error. The ground truth values for these errors were obtained through FEM with 2^{12} equidistant discretization points on each edge. The outcomes for the scenario where $v_e \equiv 1$ are presented in Table 3, while the results for the case of constant but varying v_e are displayed in Table 4.

Graph	L^2 error	L^2 error/edge	Max norm	N-K error	#Iter
Star	6.6851e-04	2.2284e-04	5.6736e-04	-3.6716e-05	7
Ladder	8.3662e-02	2.8075e-04	6.7209e-03	-1.9379e-05	26
Turan	3.8753e-02	6.1513e-04	6.1693e-03	-2.6996e-05	15
Paley	5.7848e-02	4.0738e-04	6.7808e-03	6.9660e-05	12
Barabasi	1.0297e-01	5.2536e-04	9.9284e-02	2.2018e-04	325

Table 3. Error rates of PINN method for dataset with $v_e \equiv 1$.

Graph	L^2 error	L^2 error/edge	Max norm	N-K error	#Iter
Star	1.7795e-03	5.9317e-04	1.8858e-03	2.4796e-05	5
Ladder	1.0436e-01	3.5020e-04	1.0193e-02	-7.8850e-05	18
Turan	7.9251e-02	1.2580e-03	1.3210e-02	-3.1075e-04	9
Paley	1.1980e-01	8.4369e-04	1.3559e-02	-2.4116e-04	11
Barabasi	1.2757e-01	1.3289e-03	6.0802e-02	1.4854e-04	242

Table 4. Error rates of PINN method for dataset with constant but varying v_e .

5 Conclusion

In conclusion, this study explored the application of Physics-Informed Neural Networks (PINNs) for solving partial differential equations (PDEs), highlighting their potential in tackling complex physical problems by integrating domain knowledge into neural networks. Through extensive experimentation, we have demonstrated that PINNs can effectively predict solutions to PDEs while satisfying boundary conditions and enforcing physical constraints, building on our prior work on this topic [15].

We introduced a transferable PINN-based framework for solving quantum graphs with previously unseen structures. This framework uses a nonoverlapping Neumann-Neumann substructuring method, where an arbitrary quantum graph is decomposed into its edges. The resulting subproblems are solved using one-dimensional PINNs. Notably, our approach does not rely on explicit use of the graph structure during training, allowing pretrained models to be applied to virtually any graph structure. The continuity and Neumann-Kirchhoff vertex conditions are incorporated implicitly through the Neumann-Neumann iteration, rather than directly in the loss function.

However, a key limitation of our approach is the assumption of constant coefficients. This simplification is necessary at this stage because our current models struggle to effectively learn variable coefficients represented by Gaussian Random Fields (GRFs).

Several promising directions for future research include:

- **Hyperparameter optimization:** Systematic tuning of hyperparameters to enhance model performance and convergence rates.
- **Robust pipeline development:** Creating a robust pipeline for streamlined experimentation and artifact versioning, thereby improving reproducibility and iterative enhancement.
- **Adaptive point allocation:** Implementing techniques for adaptive point allocation to focus computational resources where resolution is most needed.
- **Data aggregation:** Aggregating all the data into a single input array, enabling one full

iteration of the Neumann-Neumann method to be computed with a single pass through each model. This could potentially improve inference time.

In our future work, we aim to eliminate the assumption of constant coefficients, thereby generalizing our PINN framework to accommodate a broader class of coefficient functions. Additionally, we plan to explore alternative methods such as DeepONet [31, 49], Fourier Neural Operators (FNO) [26], Laplace Neural Operators (LNO) [11], and Kolmogorov-Arnold Networks (KAN) [29, 46]. These approaches may provide complementary benefits and enhance our ability to model complex phenomena more effectively.

References

- [1] M. Aizenman, R. Sims, and S. Warzel. Fluctuation based proof of the stability of ac spectra of random operators on tree graphs, 2005.
- [2] S. Alexander. Superconductivity of networks. a percolation approach to the effects of disorder. *Physical Review B*, 27(3):1541–1557, 1983.
- [3] C. Amovilli, F. Leys, and N. March. Electronic energy spectrum of two-dimensional solids and a chain of c atoms from a quantum network model. *Journal of Mathematical Chemistry*, 36:93–112, 06 2004.
- [4] M. Arioli and M. Benzi. A finite element method for quantum graphs. *IMA Journal of Numerical Analysis*, 38(3):1119–1163, 06 2017.
- [5] I. Babuska. Über schwarzsche algorithmen in partielle differentialgleichungen der mathematischen physik. *ZAMM*, 37(7/8):243–245, 1957.
- [6] G. Berkolaiko and P. Kuchment. *Introduction to Quantum Graphs*. Mathematical surveys and monographs. American Mathematical Society, 2013.
- [7] L. Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.
- [8] D. Bolin, M. Kovács, V. Kumar, and A. B. Simas. Regularity and numerical approximation of fractional elliptic differential equations on compact metric graphs, 2023.
- [9] J.-F. Bourgat, R. Glowinski, P. Tallec, and M. Vidrascu. Variational formulation and algorithm for trace operator in domain decomposition calculations. In *Second international symposium on domain decomposition methods for partial differential equations*. SIAM, 1989.
- [10] J.-F. Bourgat, R. Glowinski, P. Tallec, and M. Vidrascu. Analysis and test of a local domain decomposition preconditioner. In *Fourth international symposium on domain decomposition methods for partial differential equations*. SIAM, 1991.
- [11] Q. Cao, S. Goswami, and G. E. Karniadakis. Lno: Laplace neural operator for solving differential equations, 2023.
- [12] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. *Acta Numerica*, 3:61–143, 1994.
- [13] Y. Chen, L. Lu, G. E. Karniadakis, and L. D. Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express*, 28(8):11618–11633, Apr 2020.

- [14] H. Cho, K. Ayers, L. de Pills, Y.-H. Kuo, J. Park, A. Ranudskaya, and R. Rockne. Modelling acute myeloid leukaemia in a continuum of differentiation states. *Letters in Biomathematics*, 5:69–98, 2018.
- [15] M. K. Csongor L. Laczkó, Mihály A. Vággy. A transferable pinn-based method for quantum graphs with unseen structure. Manuscript accepted to 11th Vienna International Conference on Mathematical Modelling MATHMOD 2025.
- [16] Q. Dihn, R. Glowinski, and J. Périaux. *Solving elliptic problems by domain decomposition methods with applications*, page 395–426. Elsevier, 1984.
- [17] M. Dryja and O. Widlund. *An additive variant of the Schwarz alternating method for the case of many subregions*. Technical Report 339, Ultracomputer Note 131. Department of Computer Science, Courant Institute, 1987.
- [18] C. Flesia, R. Johnston, and H. Kunz. Strong localization of classical waves: A numerical study. *Europhysics Letters (EPL)*, 3(4):497–502, 1987.
- [19] C. Flesia, R. Johnston, and H. Kunz. Localization of classical waves in a simple model. *Physical Review A*, 40(7):4011–4018, 1989.
- [20] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379:113741, 2021.
- [21] P. J. F. Harris. *Carbon Nanotubes and Related Structures: New Materials for the Twenty-first Century*. Cambridge University Press, 1999.
- [22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2017.
- [23] E. L. Korotyaev and A. A. Kutsenko. Zigzag nanoribbons in external electric and magnetic fields, 2010.
- [24] N. Kovachki, S. Lanthaler, and S. Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22(290):1–76, 2021.
- [25] M. Kovács and M. A. Vággy. Neumann-Neumann type domain decomposition of elliptic problems on metric graphs. *arXiv:2402.05707*, 2024.
- [26] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.
- [27] P.-L. Lions. On the Schwarz alternating method. I. In *First international symposium on domain decomposition methods for partial differential equations*. SIAM, 1988.
- [28] P.-L. Lions. On the Schwarz alternating method. II. In *Second international symposium on domain decomposition methods for partial differential equations*. SIAM, 1989.
- [29] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark. Kan: Kolmogorov-arnold networks, 2024.
- [30] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv:1711.05101*, 2019.

- [31] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, Mar. 2021.
- [32] T. Mathew. *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations*, volume 61. Springer, Berlin, Heidelberg, 01 2008.
- [33] D. Morgenstern. Begründung des alternierenden verfahrens durch orthogonalextrapolation. *ZAMM*, 36:7–8, 1956.
- [34] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [35] Y. Saad and M. H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.
- [36] H. A. Schwarz. *Gesammelte Mathematische Abhandlungen*, volume 2. Springer Berlin Heidelberg, 1870. First published in *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 1870.
- [37] I. Simkó, C. Fábri, and A. G. Császár. Quantum-chemical and quantum-graph models of the dynamical structure of CH_5^+ . *Journal of Chemical Theory and Computation*, 19(1):42–50, 2022.
- [38] B. Smith, P. E. Bjorstad, and W. Gropp. *Domain decomposition*. Cambridge University Press, 1996.
- [39] J. D. Smith, Z. E. Ross, K. Azizzadenesheli, and J. B. Muir. HypoSVI: Hypocentre inversion with Stein variational inference and physics informed neural networks. *Geophysical Journal International*, 228(1):698–710, 08 2021.
- [40] S. L. Sobolev. L’algorithme de schwarz dans la théorie de l’élasticité. *Comptes rendus doklady de l’académie des sciences de l’URSS*, 4(13):243–246, 1936.
- [41] J. Stiasny, G. S. Misyris, and S. Chatzivasileiadis. Physics-informed neural networks for non-linear system identification for power system dynamics. In *2021 IEEE Madrid PowerTech*, pages 1–6, 2021.
- [42] P. Tallec, Y. Roeck, and M. Vidrascu. Domain decomposition methods for large linearly elliptic three-dimensional problems. *Journal of Computational and Applied Mathematics*, 34(1):93–117, February 1991.
- [43] A. Toselli and O. Widlund. *Domain Decomposition Methods – Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer, 2005.
- [44] H. A. van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.
- [45] H. Wang, R. Planas, A. Chandramowliswaran, and R. Bostanabad. Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains. *Computer Methods in Applied Mechanics and Engineering*, 389:114424, 2022.

- [46] Y. Wang, J. Sun, J. Bai, C. Anitescu, M. S. Eshaghi, X. Zhuang, T. Rabczuk, and Y. Liu. Kolmogorov arnold informed neural network: A physics-informed deep learning framework for solving forward and inverse problems based on kolmogorov arnold networks, 2024.
- [47] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34(4):581–613, 1992.
- [48] L. Yang, D. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1):A292–A317, 2020.
- [49] Y. Yang. DeepONet for Solving PDEs: Generalization Analysis in Sobolev Training, 2024.
- [50] Y. Zhao and M. L. Pasini. A deep learning approach to solve forward differential problems on graphs, 2022.

Appendix

AI tools utilized

To enhance productivity and streamline the development of this work, the following AI-powered tools were used:

- **GitHub Copilot:** This tool helped speed up the coding process by providing suggestions and generating boilerplate code during software development.
- **JetBrains Grazie:** Used for advanced grammar and spell checking, this tool ensured the linguistic quality of the document.