



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

Downloaded from: <https://research.chalmers.se>, 2026-04-11 02:46 UTC

Citation for the original published paper (version of record):

Laczkó, C., Vághy, M., Kovacs, M. (2025). A transferable PINN-based method for quantum graphs with unseen structure. *IFAC-PapersOnLine*, 59(1): 67-72.

<http://dx.doi.org/10.1016/j.ifacol.2025.03.013>

N.B. When citing this work, cite the original published paper.

A transferable PINN-based method for quantum graphs with unseen structure^{*}

Csongor L. Laczkó^{*} Mihály A. Vággy^{*} Mihály Kovács^{*,**,***}

^{*} Faculty of Information Technology and Bionics, Pázmány Péter Catholic University, H-1444 Budapest, Hungary

^{**} Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, SE-41296 Gothenburg, Sweden

^{***} Department of Analysis and Operations Research, Budapest University of Technology and Economics, Műegyetem rkp. 3-9, H-1111 Budapest, Hungary

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 adaptability across various network topologies. By incorporating edge-wise surrogates, this approach achieves experimental results comparable to those obtained with FEM across diverse network configurations.

Copyright © 2025 The Authors. This is an open access article under the CC BY-NC-ND license (<https://creativecommons.org/licenses/by-nc-nd/4.0/>)

Keywords: Physics Informed Neural Networks, partial differential equations, quantum graphs, transferable deep learning, scientific machine learning

1. INTRODUCTION

Quantum graphs, mathematical abstractions representing networks of quasi-one-dimensional systems coupled at vertices, have emerged as powerful tools for studying a diverse array of physical phenomena, such as superconductivity in granular materials (Alexander, 1983), wave guide networks (Flesia et al., 1987, 1989), cell differentiation (Cho et al., 2018) or quantum transport processes (Simkó et al., 2022). These structures provide a versatile framework for modelling complex systems with varying structural arrangements.

In recent years, there has been a growing interest in developing efficient numerical techniques for solving quantum graph problems, especially when its size renders traditional approaches obsolete through memory limitations. For classical problems on domains an overlapping decomposition was introduced in (Schwarz, 1870) more than 150 years ago, further developed in (Babuska, 1957; Morgenstern, 1956; Sobolev, 1936). Then nonoverlapping decompositions were introduced due to their inherent parallelism and the growth of high-performance computing (HPC) (Dryja and Widlund, 1987; Lions, 1988, 1989). We use a Neumann-Neumann method, originally introduced in (Bourgat et al., 1989, 1991; DiHN et al., 1984; Tallec et al., 1991), and generalized for quantum graphs in (Kovács and Vággy, 2024). For surveys on domain decomposition

methods, we refer to (Chan and Mathew, 1994; Xu, 1992). A more thorough theoretical background and historical overview can be found in (Mathew, 2008; Toselli and Widlund, 2005).

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 (Raissi et al., 2019; Kovachki et al., 2021; Cao et al., 2024; Liu et al., 2024). In particular, Physics-Informed Neural Networks (PINNs) have been designed to solve complex physical problems by integrating prior knowledge into the learning process (Raissi et al., 2019). 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 (Chen et al., 2020; Haghghat et al., 2021; Smith et al., 2021), through system identification (Stiasny et al., 2021), to the development of new classes of numerical solvers for partial differential equations (Yang et al., 2020).

While there exist PINN solutions for quantum graphs (Zhao and Pasini, 2022), they explicitly include the graph structure in the loss function to couple the loss functions on the edges. Thus, if the graph structure or the parameters change, the model has to be retrained from scratch.

^{*} Supported by the EKÖP-24-1 and the EKÖP-24-3 University Research Scholarship Programs of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund, and the K-145934 grant of the Hungarian National Research, Development and Innovation Office.

This serves as motivation for our approach, which is designed to be applicable even for unseen graph structures.

The key reference for our approach is (Wang et al., 2022), 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 formulating and solving the time-independent Schrödinger equation on quantum graphs. The following introduction is based on (Kovács and Vághy, 2024). Let $G = (V, E)$ be an arbitrary simple graph, where V represents a finite set of vertices and E represents the edges connecting them. 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 a product function space of square integrable functions on the edges. Let $u_e(v)$ denote the value of u at $v \in V$ along the edge $e \in E$.

We define $c_e(x)$ as a positive Lipschitz continuous function. Since $c_e(x)$ may not be differentiable, we transform our equation in a weak form. Furthermore, we define $v_e(x)$ as a bounded function that satisfies $v_e(x) \geq v_0$ for some $v_0 > 0$. We also assume that $f_e(x)$ is a finite square integrable function.

To define the vertex conditions, let us denote by E_v the set of edges incident to the vertex $v \in V$, and by $d_v = |E_v|$ the degree of $v \in V$. We denote by $\text{int}(G)$ the set of vertices with degree $d_v > 1$ and by ∂G the set $V \setminus \text{int}(G)$. We seek solutions that are continuous on G and satisfy the Neumann-Kirchhoff conditions, formulated as follows:

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

If $d_v = 1$, then this reduces to the classical zero Neumann boundary condition.

Here, the derivatives are assumed to be taken in the directions away from the vertex and into the edge, commonly referred to as the outgoing directions, a convention we maintain throughout this work.

In order to write the vertex conditions more compactly, let us define the vector of function values at $v \in V$ as

$$U(v) = (u_e(v))_{e \in E_v} \in \mathbb{R}^{d_v}$$

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

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

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

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

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

Then the quantum graphs can be formally written as

$$\begin{aligned} -(c_e u'_e)'(x) + v_e(x) u_e(x) &= f_e(x), \quad x \in (0, \ell_e), \quad e \in E, \\ 0 &= I_v U(v), \quad v \in \text{int}(G), \\ 0 &= C(v)^\top U'(v), \quad v \in V. \end{aligned} \quad (1)$$

For the sake of simplicity, in this paper we assume that $c_e(x) \equiv 1$ and $v_e(x) \equiv 1$. Without loss of generality we also assume that each $\ell_e = 1$, since otherwise we could rescale the equations.

2. PRELIMINARIES

2.1 Finite Element Method for quantum graphs

In this section we highlight some key facts about the Finite Element Method (FEM) for quantum graphs based on (Arioli and Benzi, 2017).

We apply the classical one-dimensional FEM to each edge with step size h_e . As the solution needs to be continuous across neighbouring vertices, we introduce special finite elements to the vertices, as shown on Figure 1.

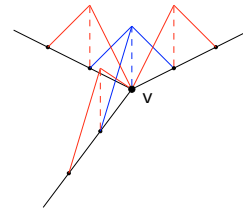


Fig. 1. Blue: special finite element for vertices. Red: classical finite elements on the edges.

After substituting the finite elements into the weak form of (1) we obtain a stiffness matrix \mathbf{H} of the following structure:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{12}^\top & \mathbf{H}_{22} \end{bmatrix}.$$

Here \mathbf{H} is symmetric and positive definite. The matrix \mathbf{H}_{11} is a block diagonal matrix, where each block corresponds to the stiffness matrix of an edge. The matrix \mathbf{H}_{22} is diagonal. The matrix \mathbf{H}_{12} contains the cross-effects of edge and vertex finite elements, and thus its nonzero structure corresponds to the graph structure. For more details, we refer to (Arioli and Benzi, 2017; Kovács and Vághy, 2024).

In (Arioli and Benzi, 2017) and (Bolin et al., 2023) the authors show the $H^1(G)$ error of the finite element solution and the weak solution is $\mathcal{O}(h)$, where $h = \max_{e \in E} h_e$, and the $L^2(G)$ error is $\mathcal{O}(h^2)$. Finally, the Neumann-Kirchhoff error of the finite element solution is also $\mathcal{O}(h)$.

2.2 Neumann-Neumann method for quantum graphs

In this section we briefly describe the Neumann-Neumann method for quantum graphs based on (Kovács and Vághy, 2024). 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 set of vertices that are shared on the boundary of

multiple subgraphs will be denoted with Γ and called the interface.

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_Γ^0 . For $n \geq 0$ the new iterate is computed as follows: first we solve the Dirichlet problems

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

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

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

Finally, we update the interface values as

$$u_\Gamma^{n+1}(v) = u_\Gamma^n(v) - \theta \sum_{e \in E_v} w_e^{n+1}(v), \quad v \in \Gamma,$$

with an appropriate $\theta \in (0, \theta_{\max})$, for some $\theta_{\max} > 0$ (Toselli and Widlund, 2005, Chapter C.3).

In (Kovács and Vághy, 2024) 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 or BiCGSTAB. However, in this paper we will use Richardson iteration, as the main goal is to test the performance of the pretrained PINNs combined with the Neumann-Neumann method on various manageable benchmark problems.

3. METHODOLOGY

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 2.2. For the sake of simplicity, 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 ∂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

We use synthetic data consisting of the right-hand side of the differential equation f_e , the values of the coefficients c_e and v_e , the numerical values for the two boundary conditions and the corresponding FEM solution.

The right-hand sides are sampled from a Whittle-Matérn Gaussian random field with parameters $\nu = 2.5$ (corresponding to twice differentiable functions) and $l = 0.1$, sampled as a vector of size $N = 128$. The exact formula for the kernel is given as:

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

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

The value of c_e and v_e are set to 1. Despite the simplification that c_e and v_e are assumed to be constant, they are implemented as input vectors of the model of size N for future works.

The dataset consists of $M = 3 \cdot 10^6$ samples generated from 1,000 boundary condition values and 3,000 right-hand side samples.

3.3 Network architecture and loss functions

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 2). 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.

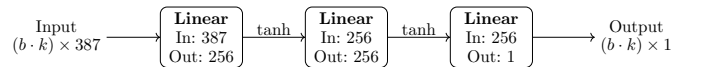


Fig. 2. Network architecture. b is the batch size, k is the number of x coordinates given as input.

The training procedure employs mini-batches, with b batch size of 512, 1024 or 2048, depending on GPU memory. Each batch of data is transformed to match the model's input format, with positional coordinate x added. The transformation utilizes NumPy's `repeat`, `expand`, and `reshape` methods. For more details, see Figure 3.

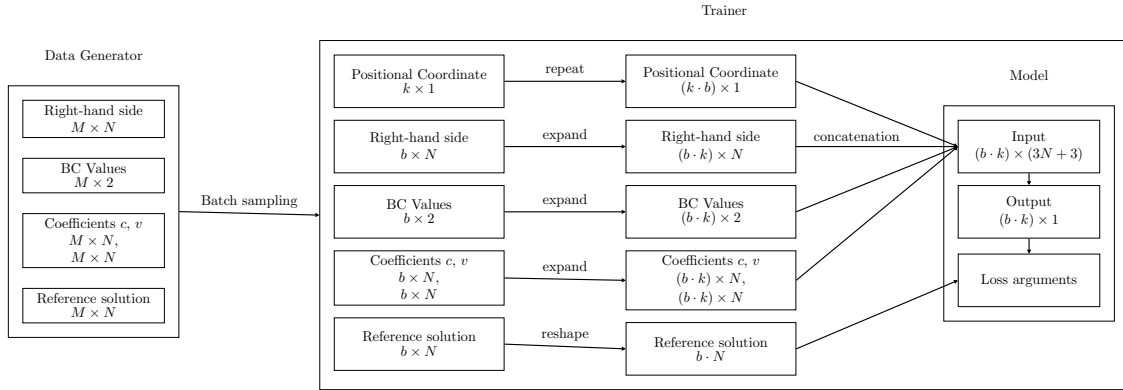


Fig. 3. Diagram illustrating the components of the code, showing the flow of data and the changes in dimensions. The functions are discretized into N points, M represents the overall size of the dataset, b is the batch size, while k is the number of positional coordinates given.

It is also important to highlight that the output of the model is the prediction at the given spatial coordinates x . During training, we use $k = N$ spatial points. But after training, the model can make 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.

The choice of an appropriate loss function plays an important role in training PINN models. The used loss function for the PINN with Dirichlet-Neumann boundary is as follows:

$$L^{DN}(\theta) = \alpha \cdot L_1(\theta) + \beta \cdot L_2(\theta) + \gamma \cdot L_3^{DN}(\theta) + \delta \cdot L_4(\theta),$$

$$L_1(\theta) = \frac{1}{N} \sum_{i=1}^N (u(x_i) - \mathcal{N}(x_i|\theta))^2,$$

$$L_2(\theta) = \frac{1}{N} \sum_{i=1}^N (-c\mathcal{N}''(x_i|\theta) + v\mathcal{N}(x_i|\theta) - f(x_i))^2,$$

$$L_3^{DN}(\theta) = (u(x_0) - \mathcal{N}(x_0|\theta))^2 + (u'(x_N) - \mathcal{N}'(x_N|\theta))^2,$$

$$L_4(\theta) = |\theta|.$$

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,
- $L_3^{DN}(\theta)$ 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

$$L_3^{DD}(\theta) = (u(x_0) - \mathcal{N}(x_0|\theta))^2 + (u(x_N) - \mathcal{N}(x_N|\theta))^2,$$

$$L_3^{NN}(\theta) = (u'(x_0) - \mathcal{N}'(x_0|\theta))^2 + (u'(x_N) - \mathcal{N}'(x_N|\theta))^2,$$

$$L_3^{ND}(\theta) = (u'(x_0) - \mathcal{N}'(x_0|\theta))^2 + (u(x_N) - \mathcal{N}(x_N|\theta))^2.$$

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.

The training process used a train-validation split ratio of 0.8, with a test dataset size equal to the validation set. The implementation was done using PyTorch with an AdamW optimizer (learning rate: 5×10^{-4} , weight decay: 1×10^{-5}). Early stopping was applied with a learning rate scheduler that reduced the rate by a factor of 0.7 if validation loss did not improve by more than 0.0001 after 15 epochs. Training stopped when the learning rate reached 1×10^{-7} or after 2000 epochs.

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, the dataset size, and the availability of the resources.

4. NUMERICAL EXPERIMENTS

In this section we briefly describe some numerical examples. 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 4 shows the structure of the used graphs.

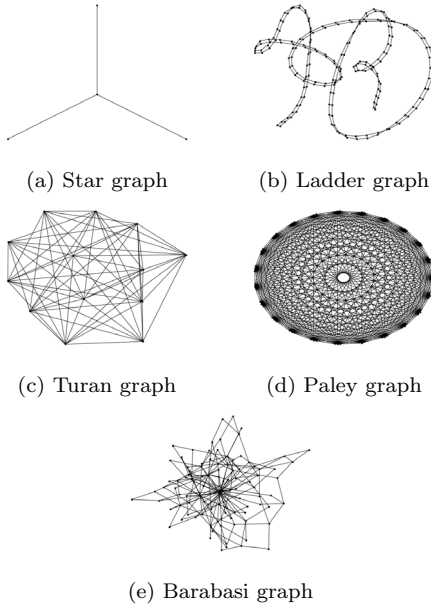


Fig. 4. Topology of graphs used in the numerical experiments

The above graphs are generated with the `networkx` package. To assess the numerical accuracy, we have measured the L^2 error, the edgewise mean L^2 error, the maximum error and the Neumann-Kirchhoff error. The ground truth for these errors is computed with FEM with 2^{12} equidistant discretization points on each edge. Table 1 shows the errors after the given number of iterations. The necessary number of iterations are essentially the same as those needed for FEM for the edge-wise surrogates.

5. DISCUSSION

In this section we discuss the limitations of our model and some future works and ideas.

5.1 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.

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 1. Error rates of PINN method in terms of L^2 error, L^2 error / edge, Neumann-Kirchhoff error and the number of iterations

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 on each edge e , although these models exhibited lower accuracy by one magnitude, especially when encountering values outside the range on which they were trained.

We thoroughly tested the number of boundary condition and right-hand side samples necessary for training. We found that a total of $3 \cdot 10^6$ samples (obtained as the Cartesian product of 1,000 boundary condition values and 3,000 right-hand sides) was sufficient to reach an accuracy that is 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.

5.2 Network architecture and loss function

We found that increasing the number and width of the linear layers also improves accuracy, again, at the cost of increased training times. Since our goal was to identify an optimal balance between accuracy and training efficiency, we avoided excessive increases in layer count and width.

We tested some other architectures, for example, the so-called linearity-preserving network of (Wang et al., 2022), 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 , γ 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.

5.3 Richardson iteration

We found that the convergence rate of the Richardson iteration is extremely sensitive to 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 algorithm to avoid the tedium of finding an optimal θ value.

6. CONCLUSION

We introduced a transferable PINN-based framework for solving quantum graphs with unseen structure. The basis of our approach is a nonoverlapping Neumann-Neumann substructuring method. We decompose an arbitrary quantum graph into its edges and solve the subproblems arising from the Neumann-Neumann method with one-dimensional PINNs. We highlight that our algorithm does not explicitly use the graph structure during training, and thus the pretrained models can be used for virtually arbitrary graph structures. In particular, the continuity and Neumann-Kirchhoff vertex conditions are not explicitly used in the loss function, only implicitly in the Neumann-Neumann iteration. The code developed for this study is available upon request.

In future works we aim to omit our assumption that the coefficient functions are constant and generalize our PINN for a reasonable space of coefficient functions. Furthermore, we wish to test different approaches, such as DeepONet, Fourier Neural Operator, Laplace Neural Operator or Kolmogorov-Arnold Network.

REFERENCES

- Alexander, S. (1983). Superconductivity of networks. a percolation approach to the effects of disorder. *Physical Review B*, 27(3), 1541–1557.
- Arioli, M. and Benzi, M. (2017). A finite element method for quantum graphs. *IMA Journal of Numerical Analysis*, 38(3), 1119–1163.
- Babuska, I. (1957). Über Schwarzsche Algorithmen in partiellen Differentialgleichungen der mathematischen Physik. *ZAMM*, 37(7/8), 243–245.
- Bolin, D., Kovács, M., Kumar, V., and Simas, A. (2023). Regularity and numerical approximation of fractional elliptic differential equations on compact metric graphs. *Mathematics of Computation*, 93(349), 2439–2472. doi:10.1090/mcom/3929.
- Bourgat, J.F., Glowinski, R., Tallec, P., and Vidrascu, M. (1989). Variational formulation and algorithm for trace operator in domain decomposition calculations. In *Second international symposium on domain decomposition methods for partial differential equations*. SIAM.
- Bourgat, J.F., Glowinski, R., Tallec, P., and Vidrascu, M. (1991). Analysis and test of a local domain decomposition preconditioner. In *Fourth international symposium on domain decomposition methods for partial differential equations*. SIAM.
- Cao, Q., Goswami, S., and Karniadakis, G.E. (2024). Laplace neural operator for solving differential equations. *Nature Machine Intelligence*, 6(6), 631–640.
- Chan, T.F. and Mathew, T.P. (1994). Domain decomposition algorithms. *Acta Numerica*, 3, 61–143.
- Chen, Y., Lu, L., Karniadakis, G.E., and Negro, L.D. (2020). Physics-informed neural networks for inverse problems in nanooptics and metamaterials. *Opt. Express*, 28(8), 11618–11633.
- Cho, H., Ayers, K., de Pills, L., Kuo, Y.H., Park, J., Ranudskaya, A., and Rockne, R. (2018). Modelling acute myeloid leukaemia in a continuum of differentiation states. *Letters in Biomathematics*, 5, 69–98.
- Dihn, Q., Glowinski, R., and Périaux, J. (1984). Solving elliptic problems by domain decomposition methods with applications. In *Elliptic Problem Solvers*, 395–426. Elsevier.
- Dryja, M. and Widlund, O. (1987). *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.
- Flesia, C., Johnston, R., and Kunz, H. (1987). Strong localization of classical waves: A numerical study. *Europhysics Letters (EPL)*, 3(4), 497–502.
- Flesia, C., Johnston, R., and Kunz, H. (1989). Localization of classical waves in a simple model. *Physical Review A*, 40(7), 4011–4018.
- Haghighat, E., Raissi, M., Moure, A., Gomez, H., and Juanes, R. (2021). A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379, 113741.
- Kovachki, N., Lanthaler, S., and Mishra, S. (2021). On Universal Approximation and Error Bounds for Fourier Neural Operators. *Journal of Machine Learning Research*, 22(290), 1–76.
- Kovács, M. and Vághy, M.A. (2024). Neumann-Neumann type domain decomposition of elliptic problems on metric graphs. *arXiv:2402.05707*.
- Lions, P.L. (1988). On the Schwarz alternating method. I. In *First international symposium on domain decomposition methods for partial differential equations*. SIAM.
- Lions, P.L. (1989). On the Schwarz alternating method. II. In *Second international symposium on domain decomposition methods for partial differential equations*. SIAM.
- Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T.Y., and Tegmark, M. (2024). KAN: Kolmogorov-Arnold Networks. *arXiv:2404.19756*.
- Mathew, T. (2008). *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations*, volume 61. Springer, Berlin, Heidelberg.
- Morgenstern, D. (1956). Begründung des alternierenden Verfahrens durch Orthogonalprojektion. *ZAMM*, 36, 7–8.
- Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). 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.
- Schwarz, H.A. (1870). *Gesammelte Mathematische Abhandlungen*, volume 2. Springer Berlin Heidelberg. First published in *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 1870.
- Simkó, I., Fábri, C., and Császár, A.G. (2022). Quantum-chemical and quantum-graph models of the dynamical structure of ch5+. *Journal of Chemical Theory and Computation*, 19(1), 42–50.
- Smith, J.D., Ross, Z.E., Azzizadenesheli, K., and Muir, J.B. (2021). HypoSVI: Hypocentre inversion with Stein variational inference and physics informed neural networks. *Geophysical Journal International*, 228(1), 698–710.
- Sobolev, S.L. (1936). 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.
- Stiasny, J., Misyris, G.S., and Chatzivasileiadis, S. (2021). Physics-informed neural networks for non-linear system identification for power system dynamics. In *2021 IEEE Madrid PowerTech*, 1–6.
- Tallec, P., Roeck, Y., and Vidrascu, M. (1991). Domain decomposition methods for large linearly elliptic three-dimensional problems. *Journal of Computational and Applied Mathematics*, 34(1), 93–117.
- Toselli, A. and Widlund, O. (2005). *Domain Decomposition Methods – Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer.
- Wang, H., Planas, R., Chandramowlishwaran, A., and Bostanabad, R. (2022). Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains. *Computer Methods in Applied Mechanics and Engineering*, 389, 114424.
- Xu, J. (1992). Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34(4), 581–613.
- Yang, L., Zhang, D., and Karniadakis, G.E. (2020). Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1), A292–A317.
- Zhao, Y. and Pasini, M.L. (2022). A deep learning approach to solve forward differential problems on graphs. *arXiv:2210.03746*.