

# An Iterative Algorithm for Neural Network Approximation to Partial Differential Equations Using Nonoverlapping Subdomain Partitions

Hyea Hyun Kim<sup>[0000-0001-7978-5602]</sup> and Hee Jun Yang<sup>[0000-0003-3628-4562]</sup>

## 1 Introduction

Neural network approximation to partial differential equations has been introduced recently and showed promising results in many application problems by employing specially designed loss functions related to the differential equations as in PINN (Physics Informed Neural Networks), deep Ritz, and deep Galerkin formulations, see [7, 12, 8]. Compared to classical approximation methods, these new approaches use a neural network to approximate the solution of the differential equations and the parameters in the neural network are trained to minimize a specially designed loss function in order to satisfy the provided differential equations with boundary or initial conditions. The advantages are they can be implemented easily without meshing the problem domain of a higher dimensional case or a complicated geometry case, and inverse problems can be handled easily by integration of the provided data to the loss function. On the other hand, the longer computation time for training parameters and larger errors in the trained solution are major limitations compared to classical approximation methods.

In the authors' previous study [11], iterative algorithms were developed for training neural network solutions where the problem domain is partitioned into overlapping subdomains, neural network functions are introduced to approximate the solution in each subdomain, and their parameters are trained for the corresponding local problem in each subdomain. The partitioned neural networks allow more flexible hyperparameters and they can reduce errors and computation time in neural network approximation as seen in experimental results in [2, 6, 11] compared to

---

Hyea Hyun Kim

Department of Applied Mathematics and Institute of Natural Sciences, Kyung Hee University, Korea., e-mail: hhkim@khu.ac.kr

Hee Jun Yang

Department of Applied Mathematics and Institute of Natural Sciences, Kyung Hee University, Korea.,e-mail: yhj109@khu.ac.kr

single neural network approximation. In [11], iterative algorithms were proposed for training neural network solutions and their convergence was analyzed by using well-developed additive Schwarz domain decomposition methods, see [10] for general introductions to domain decomposition methods. We also note that there have been developed similar iterative algorithms for neural network approximations based on alternating Schwarz methods [4, 5].

In this paper, as a continuation of our previous work in [11], we propose an iterative algorithm for neural network solutions based on non-overlapping subdomain partitions. The non-overlapping subdomain partitions are useful for modeling problems with discontinuous coefficients, interfaces, or multi physics, see [3].

Our iterative algorithm will be developed for the deep Ritz formulation [12] and parameters in each local neural network are trained for the loss function by the deep Ritz formulation of the local problem at each outer iteration. The outer iteration is performed on the auxiliary parameter  $\bar{u}$ , that approximates the solution on the subdomain interface  $\Gamma$ . For a given initial  $\bar{u}^{(0)}$ , the local problems are solved and their solutions are used to update the next iterate  $\bar{u}^{(1)}$  by the gradient descent method. The iteration is performed up to the stopping condition.

This paper is organized as follows. In Section 2, we introduce neural network approximation by the deep Ritz formulation for solving partial differential equations and in Section 3 we propose our one and two-level iterative algorithms for partitioned neural network approximation based on non-overlapping subdomains. In Section 4, numerical results are presented for two-dimensional model problems.

## 2 Deep Ritz formulation

In this section, we introduce a deep Ritz formulation [12] for solving the following Poisson problem with a Dirichlet boundary condition,

$$-\Delta u = f \text{ in } \Omega, \quad u = g \text{ on } \partial\Omega, \quad (1)$$

where we assume that the model problem in (1) is well-posed and the solution  $u$  exists. We then approximate the solution  $u$  in (1) by a neural network,  $U(\mathbf{x}; \theta)$ , that can be trained by minimizing the cost function  $\mathcal{J}(\theta)$  consisting of the two terms

$$\mathcal{J}(\theta) = \mathcal{J}_{X_\Omega}(\theta) + w_B \mathcal{J}_{X_{\partial\Omega}}(\theta),$$

where

$$\begin{aligned} \mathcal{J}_{X_\Omega}(\theta) &:= \frac{1}{|X_\Omega|} \sum_{\mathbf{x} \in X_\Omega} \left( \frac{1}{2} |\nabla U(\mathbf{x}; \theta)|^2 - f(\mathbf{x})U(\mathbf{x}; \theta) \right), \\ \mathcal{J}_{X_{\partial\Omega}}(\theta) &:= \frac{1}{|X_{\partial\Omega}|} \sum_{x \in X_{\partial\Omega}} (U(\mathbf{x}; \theta) - g(\mathbf{x}))^2. \end{aligned}$$

In the above,  $X_D$  denotes the collection of points chosen from the region  $D$  and  $|X_D|$  denotes the number of points in the set  $X_D$ . The cost function  $\mathcal{J}_{X_\Omega}(\theta)$  and  $\mathcal{J}_{X_{\partial\Omega}}(\theta)$  are designed so that the optimized neural network  $U(\mathbf{x}; \theta)$  satisfies the energy minimization property of the solution with the prescribed boundary condition, that is imposed by the additional penalty term,  $\mathcal{J}_{X_{\partial\Omega}}$  with a suitable weight factor  $w_B$ . As proposed in [9, 3], we can include an augmented Lagrangian term to the cost function,

$$L(\theta, \lambda) := \mathcal{J}(\theta) + \sum_{\mathbf{x} \in X_{\partial\Omega}} \lambda(\mathbf{x})(U(\mathbf{x}; \theta) - g(\mathbf{x})),$$

where Lagrange multipliers  $\lambda := (\lambda(\mathbf{x}))_{\mathbf{x} \in X_{\partial\Omega}}$  are introduced to enforce the boundary condition as constraints. The inclusion of the augmented Lagrangian term helps to accelerate the parameter training to give a more accurate trained solution, as shown in the experimental results in [3]. In our deep Ritz formulation, we thus include the augmented Lagrangian term and train our neural network solution for the cost function  $L(\theta, \lambda)$  and set  $w_B = 1$ .

### 3 An iterative algorithm based on non-overlapping subdomain partitions

For the model problem in (1), we now propose an iterative scheme for finding a neural network approximate solution, where the iteration algorithm is developed based on non-overlapping subdomain partitions,  $\{\Omega_i\}_i$ , of the domain  $\Omega$ . We employ a local neural network  $U_i(x; \theta_i)$  to approximate the solution in each subdomain  $\Omega_i$ , while  $U(x; \theta)$  in the previous section approximates the solution in the domain  $\Omega$ . We denote by  $\Gamma$  the interface of the subdomains in the partition, by  $\Gamma_i$  the part of  $\Gamma$  in each subdomain boundary, and by  $\Gamma_{i0}$  the part of  $\partial\Omega$  in each subdomain boundary.

We then consider the following local problems in each subdomain  $\Omega_i$  for the given interface value  $\bar{v}$  on  $\Gamma$ : Find equilibrium solutions  $u_i^*$ ,  $\Lambda_i^*$ , and  $\Lambda_{i0}^*$  of the functional  $\mathcal{L}_i(u_i, \Lambda_i, \Lambda_{i0}; \bar{v})$ ,

$$(u_i^*, \Lambda_i^*, \Lambda_{i0}^*) := \arg \left( \min_{u_i} \max_{\Lambda_i, \Lambda_{i0}} \mathcal{L}_i(u_i, \Lambda_i, \Lambda_{i0}; \bar{v}) \right)$$

in the corresponding function spaces,  $H^1(\Omega_i)$ ,  $H^{-1/2}(\Gamma_i)$ , and  $H^{-1/2}(\Gamma_{i0})$ , respectively, where the functional  $\mathcal{L}_i$  is defined as

$$\begin{aligned} \mathcal{L}_i(u_i, \Lambda_i, \Lambda_{i0}; \bar{v}) := & \left( \int_{\Omega_i} \frac{1}{2} |\nabla u_i|^2 - \int_{\Omega_i} f u_i \right) + \int_{\Gamma_i} (u_i - \bar{v})^2 \\ & + \int_{\Gamma_{i0}} (u_i - g)^2 + \int_{\Gamma_i} (u_i - \bar{v}) \Lambda_i + \int_{\Gamma_{i0}} (u_i - g) \Lambda_{i0}. \end{aligned} \quad (2)$$

We will denote the equilibrium solution  $u_i^*$  by  $u_i(\bar{v})$  and we define the following functional  $\mathcal{F}(\bar{v})$ ,

$$\mathcal{F}(\bar{v}) := \sum_i \mathcal{L}_i(u_i^*, \Lambda_i^*, \Lambda_{i0}^*; \bar{v}) = \sum_i \int_{\Omega_i} \left( \frac{1}{2} |\nabla u_i(\bar{v})|^2 - f u_i(\bar{v}) \right).$$

Letting  $\bar{u}$  be the trace value of the solution  $u$  on  $\Gamma$ , we can see that the functional  $\mathcal{F}(\bar{v})$  achieves its minimum at  $\bar{u}$  and  $\bar{u}$  is the unique equilibrium of  $\mathcal{F}(\bar{v})$ .

We then propose the following iterative algorithm on  $\bar{v}$  by applying a gradient descent method to the functional  $\mathcal{F}(\bar{v})$ :

**Algorithm 1:** (input:  $\bar{u}^{(0)}$ , output:  $(U_i(\mathbf{x}; \theta_i^*))_i, i = 1, \dots, N$ )

**Step 0:** Let  $\bar{u}^{(0)}$  be given on  $\Gamma$  and  $n = 0$ .

**Step 1:** Find  $\theta_i^{(n+1)}$  in  $U_i(x; \theta_i^{(n+1)})$  for

$$-\Delta u = f \text{ in } \Omega_i, \quad u = \bar{u}^{(n)} \text{ on } \Gamma_i, \quad u = g \text{ on } \Gamma_{i0}.$$

**Step 2:** Update  $\bar{u}^{(n+1)}$  on  $\Gamma$  as

$$\bar{u}^{(n+1)} = \bar{u}^{(n)} - \gamma \nabla_{\bar{u}^{(n)}} F(\bar{u}^{(n)}),$$

where  $\gamma > 0$  is a step size in the gradient descent method.

**Step 3:** Go to **Step 1** with  $n = n + 1$  or set the output as  $U_i(\mathbf{x}; \theta_i^*)$  with  $\theta_i^* = \theta_i^{(n+1)}$  if the stop condition is met.

In our algorithm, at each iteration, for the given  $\bar{u}^{(n)}$  on the interface  $\Gamma$ , we approximate the equilibrium solution  $u_i(\bar{u}^{(n)})$  with a neural network  $U_i(\mathbf{x}; \theta_i^{(n+1)})$  to minimize the cost function,

$$\begin{aligned} L_i(\theta_i, \lambda_i, \lambda_{i0}; \bar{u}^{(n)}) &:= \frac{1}{|X_{\Omega_i}|} \sum_{\mathbf{x} \in X_{\Omega_i}} \left( \frac{1}{2} |\nabla U_i(\mathbf{x}; \theta_i)|^2 - f(\mathbf{x}) U_i(\mathbf{x}; \theta_i) \right) \\ &+ \frac{1}{|X_{\Gamma_i}|} \sum_{\mathbf{x} \in X_{\Gamma_i}} (U_i(\mathbf{x}; \theta_i) - \bar{u}^{(n)}(\mathbf{x}))^2 + \frac{1}{|X_{\Gamma_{i0}}|} \sum_{\mathbf{x} \in X_{\Gamma_{i0}}} (U_i(\mathbf{x}; \theta_i) - g(\mathbf{x}))^2 \quad (3) \\ &+ \sum_{\mathbf{x} \in X_{\Gamma_i}} (U_i(\mathbf{x}; \theta_i) - \bar{u}^{(n)}(\mathbf{x})) \lambda_i(\mathbf{x}) + \sum_{\mathbf{x} \in X_{\Gamma_{i0}}} (U_i(\mathbf{x}; \theta_i) - g(\mathbf{x})) \lambda_{i0}(\mathbf{x}), \end{aligned}$$

where the functional  $\mathcal{L}_i$  in (2) is approximated by the numerical integration. We note that the parameters  $\lambda_i^{(n+1)}$  and  $\lambda_{i0}^{(n+1)}$  are found to maximize the cost function  $L_i$  with respect to  $\lambda_i$  and  $\lambda_{i0}$ . In practice, we set  $N_t$  as the maximum number of training epochs and update the parameter  $\theta_i$  using the Adam optimizer with a learning rate  $\epsilon$  and the parameters  $\lambda_i$  and  $\lambda_{i0}$  using a simple gradient ascent method with step sizes  $\alpha$  and  $\alpha_0$ , respectively, up to the  $N_t$  epochs,

$$\lambda_i(\mathbf{x}) = \lambda_i(\mathbf{x}) + \alpha (U_i(\mathbf{x}; \theta_i) - \bar{u}^{(n)}(\mathbf{x})), \quad \lambda_{i0}(\mathbf{x}) = \lambda_{i0}(\mathbf{x}) + \alpha_0 (U_i(\mathbf{x}; \theta_i) - g(\mathbf{x})),$$

where we use the previous epoch's trained parameters  $\lambda_i(\mathbf{x})$ ,  $\lambda_{i0}(\mathbf{x})$  and  $\theta_i$  on the right hand side terms. At the initial training epoch of the next iterate  $n + 1$ , we set  $\lambda_i(\mathbf{x}) = \lambda_i^{(n)}(\mathbf{x})$ ,  $\lambda_{i0}(\mathbf{x}) = \lambda_{i0}^{(n)}(\mathbf{x})$ , and  $\theta_i = \theta_i^{(n)}$ .

Since the cost function  $\mathcal{L}_i$  in (2) is approximated by  $L_i$ , in **Step 2** the functional  $\mathcal{F}(\bar{u}^{(n)})$  is approximated by  $F(\bar{u}^{(n)})$ ,

$$F(\bar{u}^{(n)}) := \sum_i L_i(\theta_i^{(n+1)}, \lambda_i^{(n+1)}, \lambda_{i0}^{(n+1)}; \bar{u}^{(n)})$$

and the gradient value is computed as

$$\nabla_{\bar{u}^{(n)}} F(\bar{u}^{(n)}) = - \sum_i \left( \frac{2}{|X_{\Gamma_i}|} \sum_{\mathbf{x} \in X_{\Gamma_i}} (U_i(\mathbf{x}; \theta_i^{(n+1)}) - \bar{u}^{(n)}(\mathbf{x})) + \sum_{\mathbf{x} \in X_{\Gamma_i}} \lambda_i^{(n+1)}(\mathbf{x}) \right).$$

The functional  $L_i$  can be obtained by approximating the integrals of  $\mathcal{L}_i$  in (2) using a Monte Carlo method or more accurate schemes, like a Gauss quadrature method. We note that in our **Algorithm 1**, data communication between local neural networks is needed only at each outer iteration, while in [2, 6] it is needed at every training epoch. Therefore, our algorithm reduces the communication cost in parallel computation.

To improve the scalability of Algorithm 1, we include a coarse neural network  $U_0(x; \theta_0)$  to propose the following two-level algorithm:

**Algorithm 2:** (input:  $\bar{u}^{(0)}$ , output:  $(U_i(\mathbf{x}; \theta_i^*))_i, i = 0, \dots, N$ )

**Step 0:** Let  $\bar{u}^{(0)}$  be given on  $\Gamma$  and  $n = 0$ .

**Step 1-0:** Find  $\theta_0^{(n+1)}$  in  $U_0(x; \theta_0^{(n+1)})$  for the coarse problem solution  $w_0$ :

$$-\Delta(w_0(\mathbf{x}) + U_i(\mathbf{x}; \theta_i^{(n)})) = f \text{ in } \forall \Omega_i \subset \Omega, \quad w_0 = 0 \text{ on } \partial\Omega.$$

**Step 1-1:** Find  $\theta_i^{(n+1)}$  in  $U_i(x; \theta_i^{(n+1)})$  for the local problem solution  $w_i$ :

$$\begin{aligned} -\Delta(w_i(\mathbf{x}) + U_0(\mathbf{x}; \theta_0^{(n+1)})) &= f \text{ in } \Omega_i, \\ w_i(\mathbf{x}) + U_0(\mathbf{x}; \theta_0^{(n+1)}) &= \bar{u}^{(n)} \text{ on } \Gamma_i, \quad w_i(\mathbf{x}) = g \text{ on } \Gamma_{i0}. \end{aligned}$$

**Step 2:** Update  $\bar{u}^{(n+1)}$  on  $\Gamma$  as

$$\bar{u}^{(n+1)} = \bar{u}^{(n)} - \gamma \nabla_{\bar{u}^{(n)}} F(\bar{u}^{(n)}),$$

where  $\gamma > 0$  is a step size in the gradient descent method.

**Step 3:** Go to **Step 1-0** with  $n = n+1$  or set the output as  $U_i(\mathbf{x}; \theta_i^*)$  with  $\theta_i^* = \theta_i^{(n+1)}$  if the stop condition is met.

## 4 Numerical results

In this section, we present numerical results of the proposed **Algorithm 1** and **Algorithm 2**. We first consider the Poisson model problem (1) with  $f$  and  $g$  to give

the exact solution

$$u(x, y) = \sin(\pi x) \sin(\pi y) \quad (4)$$

and with  $\Omega$  as a unit square domain. For the test example, we set the hyperparameter settings as listed in Table 1, where the domain is partitioned into  $N \times N$  uniform squares, and the total sum of the number of local network parameters and the total sum of the number of local training sample points are comparable for all the subdomain partition cases. For the local and coarse networks, we employed a fully connected network with a sine activation function. Since the subdomains are square, we approximate the integrals in  $\Omega_i$ ,  $\Gamma_i$ , and  $\Gamma_{i0}$  with a Gauss quadrature to obtain the training sample points. For the Gauss quadrature case, the additional weight factor  $w(\mathbf{x})$  is also included in the loss function  $L_i$  in (3) at each corresponding sample point  $\mathbf{x}$ .

When training the parameter  $\theta_i$ , we use the Adam optimizer with the learning rate  $\epsilon = 0.001$  and for the parameters  $\lambda_i$  and  $\lambda_{i0}$  we simply use the gradient ascent method with the step size  $\alpha = 0.1$  and  $\alpha_0 = 1$ , respectively. We perform these parameter updates up to  $N_t = 1000$  epochs per outer iteration and with  $N_o = 50$  outer iterations. In the gradient update step, we set the step size  $\gamma$  adaptively by using the Barzilai–Borwein method [1] with the initial value  $\gamma^{(0)} = 1$ . Our computation is performed on Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz and Quadro RTX 5000 with Jax library using jit compiler in python and jax.vmap to vectorize the local loss computations.

In Table 2, we report the relative errors of the trained solutions  $U_i(\mathbf{x}; \theta_i^*)$  to the exact solution and the computation time, where we perform the computation using five different random initializations on  $\theta_i$  and compute the average value and standard deviation for the corresponding trained solutions. For all the subdomain partition cases, we obtained trained solutions with a good accuracy under our settings. As increasing the number of subdomains, the error convergence gets slower in **Algorithm 1** and with the inclusion of a coarse component the error convergence becomes more robust in **Algorithm 2**.

**Table 1** Hyper parameter settings for the computation results in Table 2: The number of layers, neurons, parameters, Gauss quadrature points in one direction and the number of interior and boundary training sample points for the local and coarse (last row) networks.

Local problem						
$N \times N$	Layers	Neurons	Para	Quadrature	Interior	Boundary
$2 \times 2$	4	18	1099	35	1225	140
$3 \times 3$	4	12	517	24	576	96
$4 \times 4$	4	9	307	18	324	72
$5 \times 5$	4	7	197	14	196	56
Coarse problem	2	10	151	35	1225	140

**Table 2** Results for the Poisson problem with the exact solution in (4): the average value of relative  $L^2$ -errors to the exact solution (standard deviation) and the computation time in seconds.

$N \times N$	Algorithm 1		Algorithm 2	
	$L^2$ -error (std)	Computation time	$L^2$ -error (std)	Computation time
$2 \times 2$	0.0055 (4.63E-03)	95	0.0014 (4.96e-04)	148
$3 \times 3$	0.0045 (1.79E-03)	91	0.0047 (6.23e-04)	136
$4 \times 4$	0.0068 (2.60E-03)	89	0.0068 (2.35e-03)	136
$5 \times 5$	0.0165 (6.25E-03)	89	0.0063 (4.00e-04)	133

As a second test example, we consider a more challenging  $p$ -Laplace model problem with the exact solution in the unit rectangular domain  $\Omega$ ,

$$u(x, y) = \frac{1}{4} \sum_{k=1}^4 \sin(2^k \pi x) \sin(2^k \pi y). \quad (5)$$

We note that our algorithm can be easily extended to a general minimization problem, such as the  $p$ -Laplace model problem, by replacing the energy functional in the Poisson model problem with that in the  $p$ -Laplace model problem, i.e.,

$$E(u) := \frac{1}{p} \int_{\Omega} |\nabla u|^p - \int_{\Omega} f u.$$

where the value  $p$  is set to 2 in the Poisson problem. In our computation below, we set  $p = 4$  and use the corresponding energy functional for the loss computation. To approximate the model solution accurately enough, we set the hyperparameter settings as listed in Table 3. The settings for the optimizers are the same as in the previous example except that the number of outer iterations is set to  $N_o = 500$ . In Table 4, the error and computation time results are reported for increasing the number of subdomains in the partition. For the nonlinear problem with the multi-frequency component solution, we obtained similar results as in the previous linear problem with the smooth solution. In **Algorithm 2**, similar level error results are obtained as increasing the number of subdomains with a reduced computation time.

*Acknowledgments:* The first author was supported by NRF-2022R1A2C100388511 and RS-2025-00516964..

## References

1. BARZILAI, J., AND BORWEIN, J. M. Two-point step size gradient methods. *IMA Journal of Numerical Analysis* 8, 1 (1988), 141–148.
2. JAGTAP, A. D., KHARAZMI, E., AND KARNIADAKIS, G. E. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering* 365 (2020), 113028.

**Table 3** Hyper parameter settings for the computation results in Table 4: The number of layers, neurons, parameters, Gauss quadrature points in one direction and the number of interior and boundary training sample points for the local and coarse networks (last row).

Local problem						
$N \times N$	Layers	Neurons	Para	Quadrature	Interior	Boundary
$2 \times 2$	4	35	3921	64	8100	360
$4 \times 4$	4	17	987	32	2025	180
$6 \times 6$	4	11	441	22	900	120
$8 \times 8$	4	8	249	16	529	92
$10 \times 10$	4	6	151	13	324	72
Coarse problem	2	10	151	64	8100	360

**Table 4** Results for the  $p$ -Laplace problem with the exact solution in (5) and  $p = 4$ : the average value of relative  $L^2$ -errors to the exact solution (standard deviation) and computation time in seconds.

$N \times N$	Algorithm 1		Algorithm 2	
	$L^2$ -error (std)	Computation time	$L^2$ -error (std)	Computation time
$2 \times 2$	0.0145 (4.33E-03)	3009	0.0120 (3.79E-03)	3581
$4 \times 4$	0.0165 (3.21E-03)	1728	0.0135 (2.66E-03)	2270
$6 \times 6$	0.0546 (5.00E-02)	1235	0.0118 (5.01E-03)	1882
$8 \times 8$	0.0310 (4.11E-02)	1022	0.0154 (7.08E-03)	1725
$10 \times 10$	0.1171 (3.66E-02)	952	0.0182 (1.15E-02)	1613

- JANG, D.-K., KIM, K., AND KIM, H. H. Partitioned neural network approximation for partial differential equations enhanced with Lagrange multipliers and localized loss functions. *Computer Methods in Applied Mechanics and Engineering* 429 (2024), 117168.
- LI, K., TANG, K., WU, T., AND LIAO, Q. D3M: A deep domain decomposition method for partial differential equations. *IEEE Access* 8 (2019), 5283–5294.
- LI, W., XIANG, X., AND XU, Y. Deep domain decomposition method: Elliptic problems. In *Mathematical and Scientific Machine Learning* (2020), PMLR, pp. 269–286.
- MOSELEY, B., MARKHAM, A., AND NISSEN-MEYER, T. Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *Advances in Computational Mathematics* 49, 4 (2023), 62.
- RAISSI, M., PERDIKARIS, P., AND KARNIADAKIS, G. E. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707.
- SIRIGNANO, J., AND SPILIOPOULOS, K. DGM: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* 375 (2018), 1339–1364.
- SON, H., CHO, S. W., AND HWANG, H. J. Enhanced physics-informed neural networks with augmented Lagrangian relaxation method (al-pinns). *Neurocomputing* 548 (2023), 126424.
- TOSSELLI, A., AND WIDLUND, O. *Domain decomposition methods—algorithms and theory*, vol. 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005.
- YANG, H. J., AND KIM, H. H. Iterative algorithms for partitioned neural network approximation to partial differential equations. *Computers & Mathematics with Applications* 170 (2024), 237–259.
- YU, B., ET AL. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics* 6, 1 (2018), 1–12.