SGM-PINN: Sampling Graphical Models for Faster Training of Physics-Informed Neural Networks

John Anticev
janticev@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

Ali Aghdaei
aaghdae1@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

Wuxinlin Cheng
wcheng7@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

Zhuo Feng
zfeng12@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

ABSTRACT

SGM-PINN is a graph-based importance sampling framework to improve the training efficacy of Physics-Informed Neural Networks (PINNs) on parameterized problems. By applying a graph decomposition scheme to an undirected Probabilistic Graphical Model (PGM) built from the training dataset, our method generates node clusters encoding conditional dependence between training samples. Biasing sampling towards more important clusters allows smaller mini-batches and training datasets, improving training speed and accuracy. We additionally fuse an efficient robustness metric with residual losses to determine regions requiring additional sampling. Experiments demonstrate the advantages of the proposed framework, achieving 3× faster convergence compared to prior state-of-the-art sampling methods.

1 INTRODUCTION

Over the past few decades, partial differential equation (PDE) solvers have been playing important roles in numerous compute-intensive computer-aided design (CAD) tasks, including circuit/device simulations [8], chip thermal analysis [15], computational electromagnetics (CEM) [11], computational fluid dynamics analysis (CFD) [22]. Traditional numerical PDE solvers like finite difference methods (FDM) and finite element analysis (FEA), tackle the problem through spatial discretization. However, finer discretization for better simulation accuracy slows down performance due to the super-linear complexity of existing sparse matrix solvers. These classic PDE algorithms often require finer grids, which can be very expensive in terms of both time and hardware resources, potentially taking days on powerful computing clusters.

An emerging class of physics-informed neural-network (PINN) based methods use automatic differentiation to estimate the solution of PDEs by training a neural network (NN) to minimize the residuals of the governing equations, initial/boundary conditions, and measurement data over a set of collocation points defined on a domain tailored to each problem [10, 20]. These PINN-based solvers also address problems not easily solvable by traditional methods, such as inverse or data assimilation problems, as well as real-time simulation problems [20, 24]. Additionally, and explored in this study, they can simultaneously learn the solution to a problem across parameterized geometries [10, 20]; such as varying the design, materials, or size of the fin stack of a heatsink. As a result, the PINN-based PDE solvers can achieve orders of magnitude faster performance than conventional solvers when many variations of a design are desired. For example, a trained parameterized PINN may be used as an ultra-fast surrogate model for another gradient-based optimization method which inferences the neural network for approximate solutions across thousands of variations of a design [25].

While PINNs have demonstrated exceptional performance across a variety of problems related to PDE solvers, there are many cases in which PINNs fail to converge, or do so to a trivial or inaccurate solution [7, 9, 23]. Although better convergence can be sometimes be realized with higher-density point clouds and larger batch sizes, this imposes memory constraints that tax the GPU hardware most commonly used to accelerate deep learning workloads. This leads to long training times that require the highest-end hardware, ultimately mirroring the issues with traditional PDE solvers exhibit in terms of scalability with model size and complexity.

Among the proposed solutions to improve the convergence and efficiency of PINN training are residual-based adaptive refinement (RAR) methods [16] (where additional samples are added to regions with high PDE residuals) and importance sampling (IS) methods [18] (where each mini-batch of samples is selected via a distribution proportional to the loss value at each sample in a dense dataset). These are implemented in currently available PINN solvers such as DeepXDE [16] and Modulus Sym [10]. Both methods suffer from high computational complexity and overhead associated with frequently calculating the residuals for every sample in a dense set, and can lead to poor retention of the solution on low-residual parts of the domain as training continues [5].

To address the limitations of existing PINN-based PDE solvers, we introduce an importance-sampling framework (SGM-PINN) to exploit the conditional dependence among the training samples by leveraging probabilistic graphical models (PGMs). Instead of training a PINN model using the full set of data samples (point cloud) during the training process that may involve millions of stochastic gradient descent (SGD) iterations, SGM-PINN adaptively forms smaller epochs by selecting the most representative data samples with a graph-based importance sampling strategy. Our preliminary results show that SGM-PINN can achieve up to 3× faster convergence for training (parameterized) PINNs on several large-scale PDE problems. Our contributions include:

(1) We present SGM-PINN, a scalable graph-based IS framework based on spectral graph clustering, to improve PINN

training. The key component in SGM-PINN is a low-resistance--diameter (LRD) decomposition scheme for partitioning a PGM (graph) into strongly coupled sample clusters that can be used to reduce the computational overhead of estimating the importance score across the input data.
(2) We show the proposed method allows for retaining good solution quality while reducing the batch size and the number of sample points to speed up training.
(3) We demonstrate the addition of a spectral stability metric [4] to incorporate gradient information of the loss with respect to input data to augment sample importance scoring, in order to improve parameterized PINN training.

The rest of the paper is organized as follows: Section 2 provides a brief introduction to the theoretical foundation of PINNs and IS methods; Section 3 gives an overview of the proposed SGM-PINN framework; Section 4 demonstrates experimental results to evaluate the performance of SGM-PINN, and Section 5 that concludes this work.

2 BACKGROUND

2.1 Theoretical background of PINNs.

Consider the following general form of a PDE equation under a set of boundary condition (BC) and initial condition (IC) constraints [10]:

$$\begin{aligned} \mathcal{F}_i[u](\mathbf{x}) &= f_i(\mathbf{x}), \quad \forall i \in \{1, \cdots, N_{\mathcal{F}}\}, \mathbf{x} \in \mathcal{D} \\ C_j[u](\mathbf{x}) &= g_j(\mathbf{x}), \quad \forall j \in \{1, \cdots, N_C\}, \mathbf{x} \in \partial\mathcal{D}, \end{aligned} \tag{1}$$

where $\mathcal{F}_i$ is the general differential operator, $N_{\mathcal{F}}$ (N_C) denotes the number of BCs (ICs), $\mathbf{x}$ is the set of independent variables defined over a bounded continuous domain $\mathcal{D} \subseteq \mathbb{R}^d$, $u(\mathbf{x})$ is the solution to the PDE, C_j is the constraint operator—such as the differential, linear, or nonlinear terms representing the boundary and initial conditions—and $\partial\mathcal{D}$ is a subset of the domain boundary. Consider a simple feed-forward fully-connected neural network with nonlinear activation functions, where the approximate solution $\tilde{u}(\mathbf{x})$ can be further expressed as:

$$\tilde{u}(\mathbf{x}, \theta) = \mathbf{W}_n\{\phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_1 \circ \phi_E\}(\mathbf{x}) + \mathbf{b}_n, \tag{2}$$

where n represents the number of neural network (NN) layers, $\phi_l(\mathbf{x}_i) = \sigma(\mathbf{W}_l\mathbf{x}_i + \mathbf{b}_l)$ represents the l^{th} layer of the NN, $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $\mathbf{b}_l \in \mathbb{R}^{d_l}$ are the weight and bias of the l^{th} layer, ϕ_E is an input encoding layer, $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \cdots, \mathbf{W}_n, \mathbf{b}_n\}$ is the set of the trainable parameters of the network, and σ is the nonlinear activation function. Given the approximated solution $\tilde{u}(\mathbf{x}, \theta)$, we define the following residuals for both PDE equation and boundary (initial) condition constraints:

$$\begin{aligned} r^{(i)}_{\mathcal{F}}(\mathbf{x}, \tilde{u}(\mathbf{x}, \theta)) &= \mathcal{F}_i[\tilde{u}](\mathbf{x}) - f_i(\mathbf{x}) \\ r^{(j)}_{C}(\mathbf{x}, \tilde{u}(\mathbf{x}, \theta)) &= C_j[\tilde{u}](\mathbf{x}) - g_j(\mathbf{x}). \end{aligned} \tag{3}$$

In order to train the NN model, residuals can be encoded into a constructed loss function such that $\tilde{u}(x, \theta)$ can gradually approach the true solution $u(x)$ through iteratively optimizing the network parameters θ. The loss function can be represented in the following form:

$$\begin{aligned} \mathcal{L}(\theta) = &\sum_{i=1}^{N_{\mathcal{F}}} \int_{\mathcal{D}} w^{(i)}_{\mathcal{F}}(\mathbf{x}) \| r^{(i)}_{\mathcal{F}}(\mathbf{x}, \tilde{u}(\mathbf{x}, \theta)) \|_p \, d\mathbf{x} \\ &+ \sum_{j=1}^{N_C} \int_{\partial\mathcal{D}} w^{(j)}_{C}(\mathbf{x}) \| r^{(j)}_{C}(\mathbf{x}, \tilde{u}(\mathbf{x}, \theta)) \|_p \, d\mathbf{x}, \end{aligned} \tag{4}$$

where $w^{(i)}_{\mathcal{F}}$ and $w^{(j)}_{C}$ are the weight functions to scale the corresponding loss terms. The model may be trained by stochastic gradient descent such that the model parameters are updated by:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla_\theta \mathcal{L}(\theta^{(t)}) \tag{5}$$

2.2 Importance Sampling for Training PINNs

Current State of the Art. Prior works have explored IS based on the gradient of the loss with respect to the model parameters θ [14, 18]:

Lemma 1. *Let $H^{(t)} = \nabla_\theta \mathcal{L}(\theta^{(t)})$ the gradient in Equation 5. The convergence of SGD can be accelerated by sampling the input variables from a distribution P that minimizes* $\mathbf{Trace}(\mathbb{V}_P[H^{(t)}])$*, which is accomplished when the probability $P^{(t)}_{x_i}$ of sampling any given point x_i at iteration i is*

$$P^{(t)}_{x_i} \propto \|H^{(t)}\|_2 \tag{6}$$

Additionally, it is shown in [13] that $\|H^{(t)}\|_2$ at a single sample is bounded by a linear transformation of the loss at that sample, and a sampling distribution:

$$P^{(t)}_{x_i} \propto \mathcal{L}(\theta^{(t)}) \tag{7}$$

is consistent with Equation 6 [18].

Limitations of existing IS methods. The prior IS method works effectively for training non-parameterized PINNs but may fail for training parameterized PINNs. The sampling probability in (6) and (7) only considers the gradient with respect to the neural network parameters (θ) but ignores the impact due to varying input parameters, such as the geometric variations in point clouds. Our experiments using the prior IS method for training parameterized PINNs shows poor generalization capability as discussed in Section 4.2.

3 THE PROPOSED SGM-PINN FRAMEWORK

3.1 Overview

The proposed SGM-PINN framework. In this work, we introduce a novel graph-based IS framework, SGM-PINN, for improving the speed and solution quality of the PINN training process. As shown in Figure 1, SGM-PINN aims to improve the efficiency of each training step by **(1)** forming more compact epochs on the fly, and **(2)** selecting more samples from within clusters that have been scored with high importance. The process consists of four key components: **(S1)** PGM estimation of the point cloud, **(S2)** construction of conditionally-independent clusters of nodes via LRD decomposition, **(S3)** a stability estimation scheme (for parameterized PINNs), to incorporate additional gradient information with respect to varying input parameters, and **(S4)** batch ranking and selection for SGD

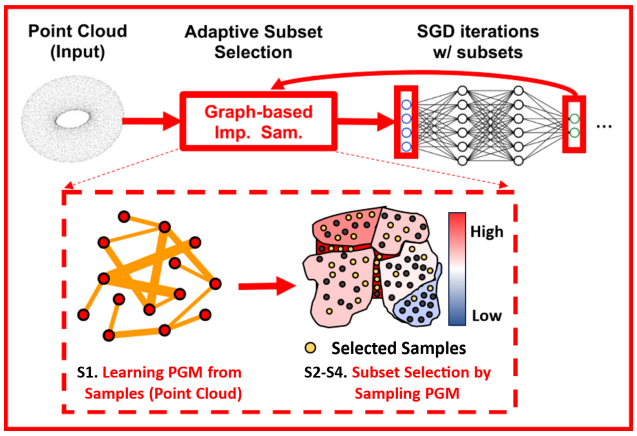

Figure 1: The SGM-PINN framework for training PINNs.

iterations. The proposed sampling strategy allows focusing on the most critical and representative data samples during each training iteration, which immediately improves the overall training process.

Advantages of our approach. There are several potential benefits of the proposed approach: **(1)** the mini-batch size for the gradient descent algorithm can be reduced, allowing for faster iterations without sacrificing solution quality; **(2)** the overall dataset size can be reduced, allowing the same models to be run on fewer GPUs or hardware with less available memory; **(3)** a small subset of samples can be used for estimating the importance scores of the dataset, decreasing the overhead compared to other IS techniques which require updated gradient calculations for every sample; and **(4)** the reduced overhead achieved by using the PGM to estimate the score for multiple samples can be easily extended to a variety of IS metrics, or combined with new metrics designed to address additional shortcomings in PINN-based solvers, as discussed in Sections 3.5 and 4.2.

3.2 Constructing PGMs from Point Cloud (S1)

A probabilistic graphical model (PGM) expresses probabilistic relationships between variables via links between nodes representing the variables [3]. In traditional PDE solvers, discretization methods operate on the principle that the solution at any given point is influenced by its immediate neighbors, with interactions between adjacent mesh or grid points playing a crucial role in problems such as FEA. In the PINN examples discussed in this paper, a randomized point cloud of N collocation points with M input features $X = \mathbb{R}^{N\times M}$ is generated prior to training as described in [20].

With the assumption that nearby points will be highly correlated in a relatively dense initial point cloud, we can construct a k-nearest-neighbor (kNN) graph using the low-dimensional spatial coordinates (x,y,z) of each point in the dataset to represent a conditional relationship between points inversely proportional to their distance. This graph, created using an existing highly-efficient kNN algorithm [17], serves as an undirected PGM for the input data X. At later stages in training this model can be re-built in parallel while incorporating additional features from the output, such as flow or temperature.

3.3 Node Clustering by LRD Decomposition (S2)

Node clustering is achieved via a low-resistance-diameter (LRD) decomposition method that allows partitioning any given PGM into multiple node (collocation point) clusters with bounded resistance diameter [1]. Since the resistance distance between two nodes on a PGM encodes the conditional dependence between the two corresponding data samples, LRD will guarantee that only the most similar data samples will be grouped into the same cluster, which thereby allows SGM-PINN to select highly-representative data samples during the training of PINNs.

Definition 3.1. *The effective resistance between nodes* $(p,q) \in |V|$ *is defined as*

$$R_{p,q}^{eff} = e_{p,q}^{\top} L^{\dagger} e_{p,q} = \sum_{i=2}^{|V|} \frac{(u_i^{\top} e_{p,q})^2}{u_i^{\top} L u_i}, \tag{8}$$

where $L^{\dagger}$ *denotes the Moore-Penrose pseudo-inverse of the graph Laplacian matrix* L, $u_i \in \mathbb{R}^{|V|}$ *for* $i = 1, ..., |V|$ *denote the unit-length, mutually-orthogonal eigenvectors corresponding to Laplacian eigenvalues* λ_i *for* $i = 1, ..., |V|$, *and* $e_p \in \mathbb{R}^{|V|}$ *denotes the standard basis vector with all zero entries except for the* p*-th entry being* 1. $e_{p,q} = e_p - e_q$.

In [2] it is proven possible to decompose a simple graph into multiple node clusters of effective-resistance diameter at most the inverse of the average node degree (up to constant losses) by removing only a constant fraction of edges. This is accomplished without significantly impacting the graph conductance (keeping the global structure of the graph intact).

Directly computing effective resistances according to Definition 3.1 is practically infeasible for large PGMs. However, as the proposed LRD method only requires approximate estimation of edge effective resistances, we use a highly-scalable (linear time) algorithm by exploiting a Krylov subspace approach, discussed in detail in [1].

Since the proposed decomposition method has a nearly-linear complexity, it can be efficiently applied to large-scale PGMs (graphs) with millions of data samples (nodes). For additional performance, we also decompose the dataset into grids and perform S1 and S2 in independent sub-processes while training continues either with uniform sampling, or a previously calculated distribution. Speedup is roughly linear with the number of available threads.

3.4 Stability Score for Parameterized PINNs (S3)

We use a black-box spectral method for assessing the robustness (stability) of ML models and data, relying on graph-based manifold representations [4], referred to here as the Inverse Stability Rating (ISR). ISR introduces the concept of Distance Mapping Distortion (DMD), denoted as $\gamma^F(p,q)$ for a node pair (p,q) transformed by a function $Y = F(X)$, $F(X)$ in this case being the NN $\tilde{u}(\mathbf{x},\theta)$. It is defined as the ratio of distances between nodes p and q on the

[1]The maximum effective-resistance distances between any two nodes within the same graph cluster will not exceed a given threshold.

output and input graphs, $\gamma^F(p,q) \triangleq \frac{d_Y(p,q)}{d_X(p,q)}$, where $d_X(p,q)$ and $d_Y(p,q)$ represent the distances between nodes p and q on the input and output graphs. Intuitively, the DMD can be employed to estimate the change in distance on the output graph (manifold) due to a perturbation on the input graph (manifold), with the largest γ^F_{max} representing the fastest change in output with respect to input.

Lemma 2. *The ISR is an upper bound of the best Lipschitz constant K^* on F(X) [4].*

$$\boldsymbol{ISR}^F \overset{def}{=} \lambda_{max}(L_Y^+ L_X) \geq K^* \geq \gamma^F_{max} \tag{9}$$

Lemma 3. *ISR defines the edge score between two nodes p, q, $\boldsymbol{ISR}^F(p,q)$ where $V_r \overset{\text{def}}{=} \left[v_1\sqrt{\lambda_1}, ..., v_r\sqrt{\lambda_r}\right]$, and λ_r, v_r represent the first r largest eigenvalues and corresponding eigenvectors of $L_Y^+ L_X$ as [4]:*

$$\boldsymbol{ISR}^F(p,q) \overset{def}{=} \|V_r^\top e_{p,q}\|_2^2 \propto \left(\gamma^F(p,q)\right)^3 \tag{10}$$

Then, individual sample stability can be quantified as the average edge score among all q_i of p, the set of which is denoted by $\mathbb{N}_X(p) \in V$ [4]:

$$\boldsymbol{ISR}^F(p) \overset{\text{def}}{=} \frac{1}{|\mathbb{N}_X(p)|} \sum_{q_i \in \mathbb{N}_X(p)} \|V_r^\top e_{p,q_i}\|_2^2 \tag{11}$$

Each edge score acts as a surrogate for the directional derivative between two nodes on $F(X)$[4], which in our application are the NN losses, such that:

$$\mathbf{ISR}^F(x_i) \geq ||\nabla_{x_i} \mathcal{L}(\theta)||_2 \tag{12}$$

In [18] additional computational efficiency is achieved by partitioning nodes among a much smaller set of random "seeds"[18], updating the loss value at each seed to calculate $\mathcal{L}(\theta^{(t)})$ and assigning that value to nearby samples in a piece-wise fashion before calculating $P^{(t)}_{x_i}$ for each sample. In our method, we also use nearby neighbors to adjust the probability of sampling points we did not directly update the loss for, but we can add additional weight to clusters with high ISR. As it represents quickly changing local losses, it implies that the loss estimate for that cluster may be poor, and potentially important points may be ignored.

3.5 Batch Ranking and Selection (S4)

Only r points in each cluster of samples are updated with the latest losses to rank that cluster relative to others, allowing a reduced number of passes through the NN to update P. Sampling is maintained with the principle in Equations 6,7 by sampling more from clusters with losses higher than other clusters, as described in Algorithm 1. There is a floor of 1 sample per cluster when determining an epoch, so no cluster is entirely ignored in any epoch. This mitigates the potential for failing to refine or 'forgetting' the solution [5] in areas with relatively low scores as training continues. For parametric examples the ISR discussed in S3 is added as an additional term to the cluster score prior to ranking, using the same subset of samples as—and normalized with—the other PDE losses. S3 is also performed on a background thread, so the only additional overhead on GPU resources and overall wall time is the $r \times N$ loss calculations every τ_e iterations.

3.6 Algorithm Flow and Complexity

Algorithm 1 The algorithm flow of SGM-PINN

Input: Sample matrix $X = \mathbb{R}^{N \times M}$ of N samples with M selected features, the ratio r of points in each cluster to sample for estimating loss. τ_e is the number of times to repeat the epoch.
Output: An epoch of mini-batches for training.

```
From X, create a kNN-graph G
Use the LRD Algorithm to split G into n_c clusters of similar samples.
S ← an array of the sizes of each cluster.
while step_count < step_target do
    S* ← r · S_i points from each cluster in S
    Calculate the losses for S*
    From S*, apply the ISR algorithm.
    L ← combined losses and ISR for each cluster
    Map L to a range of proportional sampling ratios P
    Create an epoch with of P_i · S_i samples from each cluster
    while step_count%τ_e ≠ 0 do
        Shuffle and return the epoch
    end while
    if τ_G has passed then
        Start 1,2,3 in a background task
    else if S_new ready then
        S ← S_new
    end if
end while
```

Algorithm 1 shows the key steps in SGM-PINN. Step 1 kNN construction using the HNSW algorithm [17] has a complexity of $O(N\log(N))$, where N is the number of nodes. Step 2 LRD is a nearly-linear time algorithm with a runtime proportional to the number of edges the graph G, which is determined by $N \times k$, $O(kN)$. Sampling losses for importance scores are an additional overhead cost, with the number of extra forward passes determined by $\tau_e \times r \times N$ for relatively frequent loss updates, as well as $\tau_G \times N$ for infrequent G updates. ISR has a complexity $O(N\log(N))$. This gives an overall complexity of $O(N\log(N)) + O(kN\tau_e + N\tau_G)$.

4 EXPERIMENTAL RESULTS

We implement the proposed SGM-PINN framework on Nvidia's Modulus v22.09 (previously on SIMNET v21.06) platform [10]. We conduct extensive experiments for evaluating the performance and efficiency of the proposed SGM-PINN framework. All tests were run on a Xeon Gold 6244 CPU @ 3.60GHz, 1.5TB of 2933 MHz available system memory, and 1 Tesla V100 32GB GPU. Examples are shipped with Modulus and use the default settings where applicable. An implementation is available online at https://github.com/Feng-Research.

Modulus also includes an importance sampling implementation based on [18] (labeled MIS here) which we benchmark against our method. This method assigns a sampling probability based on the 2-norm of the velocity (u,v) derivatives. For an even comparison we reduce how often the dataset is updated to match τ_e. By default MIS re-calculates sample probabilities every epoch and may perform slower than the baseline random sampling due to loss updates not contributing to training. We additionally only apply MIS to the sampling of interior points, as SGM-PINN has not yet been implemented for the boundary or initial conditions.

Table 1: Minimum Validation Errors and Time to Achieve for LDC_zeroEq. $T(M_\beta\_j)$ is the time taken by a sampling method in the top row to achieve $Min(j)$ of M_β. Blanks are left where that value was not achieved. The best and second-best results are bolded and italicized, respectively. The diagonal (time each took to get to its own best value) is underlined.

Label	U_{500}	U_{4000}	MIS_{500}	SGM_{500} (ours)
$Min(u)$	0.0879	0.0480	*0.0431*	**0.0412**
$Min(v)$	0.1169	*0.0589*	0.0623	**0.0573**
$Min(nu)$	0.2173	0.1788	*0.1735*	**0.1595**
$T(U_{4000}\_u)$	-	32.55	*16.53*	**9.48**
$T(MIS_{500}\_u)$	-	-	*24.19*	**11.26**
$T(SGM_{500}\_u)$	-	-	-	**16.91**
$T(U_{4000}\_v)$	-	*32.61*	-	**11.45**
$T(MIS_{500}\_v)$	-	*30.23*	35.29	**10.25**
$T(SGM_{500}\_v))$	-	-	-	**18.18**

The two example problems presented in this paper are simulations related to computational fluid dynamics (CFD). The first, Lid Driven Cavity (LDC) [19] is a well-studied benchmark example, with the addition of a zero-equation turbulence model. Outputs measured against OpenFOAM [12] validation data are u (x-velocity), v (y-velocity), and ν (kinematic viscosity). Further details are in Section 4.1 The second, annular ring (AR)[19] is another 2D laminar flow example that considers the flow from an inlet to an outlet through a symmetrical annular ring with a parameterized inner radius r_i. Outputs are u, v, and p (pressure), with validation results available at r_i=1.0, r_i=0.875, and r_i=0.75. All NNs use a fully connected architecture with width 512, depth 6, and SiLU[6] activation functions. Data presented is an average of 5 runs for each example.

4.1 LDC, Non-Parametric, without S3

Experimental Setup. LDC with zero-eq turbulence, the Reynolds number $Re = 1000$, and the top wall is moving at $1m/s$. U_{4000} is the 'baseline' example with uniform random sampling, batch size $\beta = 4000$, and number of samples $N = 16M$. The remaining methods U_{500}, MIS_{500}, SGM_{500} have $\beta = 500$ and $N = 8M$. MIS_{500} and SGM_{500} re-calculate sample scores (τ_e) every 7k iterations. SGM_{500} uses $r = 15\%$ samples per cluster and fully recalculates clusters (τ_G) every 25K iterations with kNN size $k = 30$ and LRD level $\mathbb{L} = 10$.

As shown in Figure 2 and Table 1, SGM-PINN improves both runtime and accuracy for the LDC example. Targeting the same accuracy as the baseline, SGM-PINN achieves a runtime improvement of 3.43× in u and a 2.85× in v. The best result is achieved 1.79× faster than the baseline's, with a 14%, 3%, and 11% reduction in relative error in u, v, and nu respectively compared to the baseline. The runtime improvement in u and v compared to the built-in importance sampling technique is 2.15× and 3.44×. Reducing the batch size to 500 and taking the total sample count to only 500,000, SGM-PINN achieves at least 2.5× speedup while reaching the same accuracy across all variables compared to the baseline. It completes 2.5 million iterations in 19 hours compared to 1 million in 25 hours

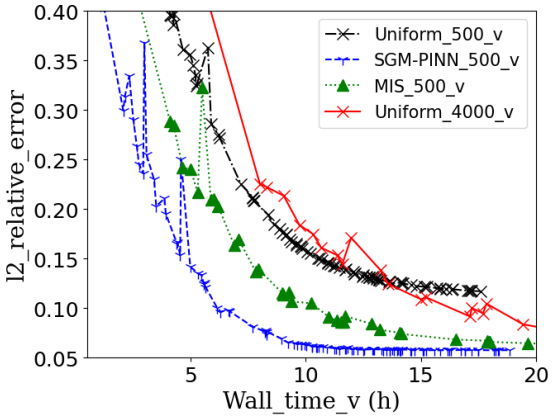

Figure 2: Solution error by wall time (lower) for v in the LDC example. $Uniform_{500}$, $SGM\text{-}PINN_{500}$, and MIS_{500} have batch sizes of 500 and a total of 500,000 collocation points. The baseline $Uniform_{4000}$ has a batch size of 4000 and 4M total collocation points.

by $Uniform_{4000}$. Without either MIS or SGM-PINN, reducing the batch and dataset size with uniform sampling cuts the final solution quality in half with no benefit to convergence speed.

4.2 Parameterized Annular Ring, with S3

Experimental Setup. Physical parameters are inlet velocity of 1.5m/s, channel width of 2m, and viscosity $\nu = 0.1$ with a parameterized interior radius of $r = [0.75, 1.1]$. U_{4096} is the 'baseline' example with uniform random sampling, batch size $\beta = 4096$, and number of samples $N = 16M$. The remaining methods U_{1024}, MIS_{1024}, SGM_{1024}, and $SGM\text{-}S_{1024}$ have $\beta = 1024$ and $N = 8M$. ($\tau_e = 7$K) for *SGM* and *MIS*. For both *SGM* methods $k = 7$, $\mathbb{L} = 6$, $r = 15\%$ and $\tau_G = 60$K. A key application for PINN solvers is the ability to solve a problem across parameterized geometries [10][21]. Both SGM-PINN and MIS have trouble with the parameterized example compared to uniform importance sampling, as shown in Figure 3 and summary Table 2. It is observed that for parameterized training SGM alone decreases performance (SGM-PINN in Figure 3). MIS additionally decreases performance, although to a lesser degree. Including the stability metric (SGM-S-PINN) in our method allows us to maintain accuracy in u and v relative to the baseline uniform sampling, while improving p by 12% (from 0.132 to 0.116) despite training for half the time. The average error for v during training is provided in Figure 3. Figure 4 visualizes the solution errors for p at 350K iterations for all methods, showing our method has the lowest error compared to validation data while taking the least time.

5 CONCLUSION

In this work, we introduce a graph-based sampling framework for speeding up the training of PINNs. SGM-PINN allows the importance score of multiple samples to be estimated via selection of highly-correlated clusters within the point cloud by leveraging PGMs. We additionally demonstrate the inclusion of a stability

Table 2: Results for Annular Ring Parameterized, averaged. The validation value for p does not monotonically decrease to a consistent value during training—it reaches a minimum well before u and v are trained and then trends upward before levelling out. Since v takes longest to converge to a minimum, the value for p is given then.

Label	U_{1024}	U_{4096}	MIS_{1024}	$SGM\text{-}S_{1024}$
$Min(u)$	0.0289	*0.0285*	0.0294	**0.0278**
$Min(v)$	0.0279	*0.0275*	0.0278	**0.0274**
p at $Min(v)$	*0.123*	0.132	0.137	**0.116**
$T(U_{1024}\_u)$	*2.20*	3.23	-	**2.00**
$T(U_{4000}\_u)$	-	*3.63*	-	**2.00**
$T(MIS_{1024}\_u)$	**1.65**	3.02	*2.32*	*2.00*
$T(SGM\text{-}S_{500}\_u)$	-	-	-	**2.14**
$T(U_{1024}\_v)$	3.00	3.41	*2.32*	**2.09**
$T(U_{4000}\_v)$	-	*6.19*	-	**2.53**
$T(MIS_{1024}\_v)$	-	3.67	*3.28*	**2.14**
$T(SGM\text{-}S_{500}\_v)$	-	-	-	**3.00**

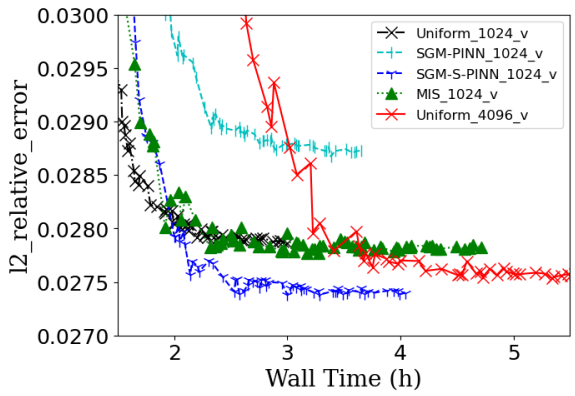

Figure 3: Solution errors of v for parameterized PINN for the AR example compared to the OpenFOAM validation data averaged at r_i=1.0,0.88, and 0.75, respectively.

score in calculating importance for improving parameterized training. Our experiments show SGM-PINN leads to a 2×-3× runtime improvement for training PINNs related to CFD problems. Future work will focus on further reducing the overhead for implementing SGM-PINN and including importance sampling on BCs. More complex examples can also be sensitive to the hyper-parameters k and $\mathbb{L}$, as is the performance overhead. Automatic tuning of these parameters would be preferred. Additionally, much larger PDE problems from more varied domains need to be tested.

6 ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Grants CCF-2212370, CCF-2205572, and CCF-2021309.

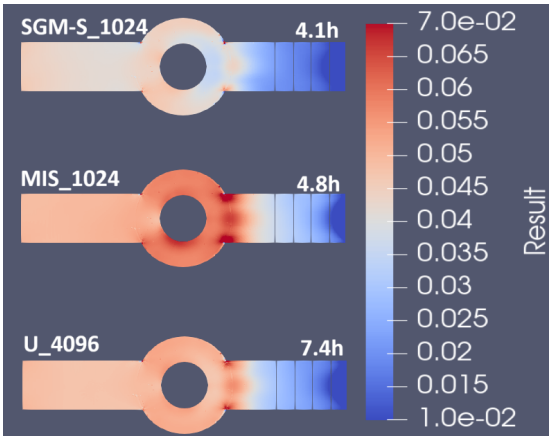

Figure 4: Visualized absolute errors for p at r_i=1.0

REFERENCES

[1] Ali Aghdaei and Zhuo Feng. 2022. HyperEF: Spectral Hypergraph Coarsening by Effective-Resistance Clustering. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[2] Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. 2018. Graph Clustering using Effective Resistance. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[3] Christopher M. Bishop. 2023. *Graphical Models*. MTM.

[4] Wuxinlin Cheng, Chenhui Deng, Zhiqiang Zhao, Yaohui Cai, Zhiru Zhang, and Zhuo Feng. 2021. SPADE: A Spectral Method for Black-Box Adversarial Robustness Evaluation. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, Vol. 139. PMLR, 1814–1824. https://proceedings.mlr.press/v139/cheng21a.html

[5] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. 2023. Mitigating propagation failures in physics-informed neural networks using retain-resample-release (r3) sampling. In *Proceedings of the 40th International Conference on Machine Learning* (, Honolulu, Hawaii, USA,) *(ICML'23)*. JMLR.org, Article 288, 39 pages.

[6] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks* 107 (2018), 3–11. https://doi.org/10.1016/j.neunet.2017.12.012 Special issue on deep reinforcement learning.

[7] Salah A Faroughi, Nikhil Pawar, Celio Fernandes, Maziar Raissi, Subasish Das, Nima K. Kalantari, and Seyed Kourosh Mahjour. 2023. Physics-Guided, Physics-Informed, and Physics-Encoded Neural Networks in Scientific Computing. arXiv:2211.07377 [cs.LG]

[8] Wolfgang Fichtner, Donald J Rose, and Randolph E Bank. 1983. Semiconductor device simulation. *SIAM J. Sci. Statist. Comput.* 4, 3 (1983), 391–415.

[9] Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. 2023. Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications. arXiv:2211.08064 [cs.LG]

[10] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. 2021. NVIDIA SimNet®: An AI-Accelerated Multi-Physics Simulation Framework. In *International Conference on Computational Science*. Springer, 447–461.

[11] Umran S Inan and Robert A Marshall. 2011. *Numerical electromagnetics: the FDTD method*. Cambridge University Press.

[12] Hrvoje Jasak. 2009. OpenFOAM: Open source CFD in research and industry. *International Journal of Naval Architecture and Ocean Engineering* 1, 2 (2009), 89–94. https://doi.org/10.2478/IJNAOE-2013-0011

[13] Angelos Katharopoulos and François Fleuret. 2017. Biased Importance Sampling for Deep Neural Network Training. *CoRR* abs/1706.00043 (2017). arXiv:1706.00043 http://arxiv.org/abs/1706.00043

[14] Angelos Katharopoulos and François Fleuret. 2018. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. In *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:3663876

[15] Peng Li, Lawrence T Pileggi, Mehdi Asheghi, and Rajit Chandra. 2004. Efficient full-chip thermal modeling and analysis. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, 319–326.

[16] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. 2021. DeepXDE: A Deep Learning Library for Solving Differential Equations. *SIAM Rev.* 63, 1 (jan 2021), 208–228. https://doi.org/10.1137/19m1274067

[17] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[18] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. 2021. Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering* 36, 8 (apr 2021), 962–977. https://doi.org/10.1111/mice.12685

[19] Nvidia 2022. *Modulus User Guide* (22.09 ed.). Nvidia. Sections: Introductory Example, Zero Equation Turbulence, Importance Sampling.

[20] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. 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.

[21] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. 2020. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering* 361 (apr 2020), 112732. https://doi.org/10.1016/j.cma.2019.112732

[22] Jiyuan Tu, Guan Heng Yeoh, and Chaoqun Liu. 2018. *Computational fluid dynamics: a practical approach.* Butterworth-Heinemann.

[23] Sifan Wang, Xinling Yu, and Paris Perdikaris. 2022. When and why PINNs fail to train: A neural tangent kernel perspective. *J. Comput. Phys.* 449 (2022), 110768. https://doi.org/10.1016/j.jcp.2021.110768

[24] Kailai Xu and Eric Darve. 2019. The Neural Network Approach to Inverse Problems in Differential Equations. arXiv:1901.07758 [math.NA]

[25] Tianju Xue, Alex Beatson, Sigrid Adriaenssens, and Ryan Adams. 2020. Amortized finite element analysis for fast PDE-constrained optimization. In *International Conference on Machine Learning*. PMLR, 10638–10647.