Hybrid Neural Network-Monte Carlo Approach for Efficient PDE Solvers

Hong Meng Yam
Department of Computer Science
Stanford University
hongmeng@stanford.edu

Ethan Hsu
Department of Computer Science
Stanford University
ethanhsu@stanford.edu

Ivan Ge
Department of Physics
Stanford University
ivange@stanford.edu

Abstract

Herein, we present a novel method that utilizes neural networks to improve solution generation from Monte Carlo Partial Differential Equation solvers. Current Monte Carlo PDE solvers, such as the Walk-On-Spheres algorithm, suffer from high variance and long computational times. To address this issue, we implement a hybrid neural network model that takes in a small set of early iterations from the Monte Carlo PDE solver for various elliptic PDEs and approximates the solution to the PDE at a much later iteration. We analyze the results from three different methods: fine-tuning the VGG-16 model, utilizing a SUNet model, and implementing a GAN. We show that the GAN performs the best and allows for the solution of any elliptic PDE and its associated forcing function to be generated on a 2D slice of an object for computer graphics applications.

1. Introduction

Solving elliptic PDEs is incredibly useful for computer graphics applications ranging from 3D reconstruction to animation techniques [6]. However, various traditional methods for geometry processing, such as finite element methods (FEM), are slow and require excessive memory and long execution times [16]. Monte Carlo methods to solve PDEs represent a new avenue of research to improve geometric modeling with PDE algorithms. While current state-of-the-art Monte Carlo PDE solvers are unbiased and discretization-free [18], they are still often computationally expensive due to high variance and result in noisy images when run at a low number of iterations.

To address the computationally expensive Monte Carlo PDE solvers, progress has been made toward utilizing neural networks to decrease the computational time to generate solutions to elliptic PDEs. Numerous works utilize various deep neural networks for fast approximation of PDE solutions [2, 9, 7], Recent advances in Physics-Informed Neural Networks (PINNs) [3] have shown great promise in solving PDEs through the incorporation of prior information, thereby reducing the likelihood of results that deviate greatly from ground truth. These networks are often fast, but can be unstable and return highly biased solutions due to the use of self-supervised loses.

There has been recent interest in exploring the possibilities of utilizing Monte Carlo methods with neural solvers to predict solutions to certain PDEs and their boundary conditions [21]. However, they are mostly using Monte Carlo methods for self-supervised training on individual PDE functions [10].

Drawing on this, we instead investigate the possibility of generalizing the approach of a hybrid Monte Carlo method and neural network solver. Our solution seeks to broadly address the computational inefficiencies of Monte Carlo PDE solvers by implementing a neural network trained on a dataset consisting of variable-coefficient elliptic PDEs and boundaries, such that at test time it is much faster than existing neural network methods.

We generate a dataset consisting of different forcing functions for the PDEs and then use a Monte Carlo Walk-On-Spheres method to generate solution instances for early iterations as well as one much later iteration. These are utilized to train the neural network to predict the later iteration based on the early iterations for each PDE. This model can then be applied to any elliptic PDE with early iterations of the unbiased, noisy solution from a Monte Carlo PDE solver to generate a final solution estimate. We hypothesize that the use of a more complex neural network architecture will allow us to generalize to all variable-coefficient elliptic PDEs without having a train a neural network for each individual PDE, resulting in a much more efficient solver.

In essence, our main contribution in this paper is a hybrid Monte Carlo-Neural Network PDE solver that is gives us a more accurate result in the same amount of computation time as compared to current Monte Carlo and Neural Network solutions, as well as achxieving a low-error solution in a much shorter time.

2. Related Work

This paper draws from and seeks to improve on existing work in Monte Carlo PDE Solvers and Neural network-based PDE solvers.

2.1. Monte Carlo PDE Solvers

Many different algorithms for implementing Monte Carlo methods to solve PDEs have been proposed in the past few decades. The Walk on Spheres (WoS) algorithm remains the most widely used [13], consisting of taking random walks on a grid by using a sphere as the boundary at which the next step would be taken. In other words, if we were to start at a point x_0 on the grid, the next step x_1 would be any point randomly taking from a sphere centered at x_0 with a radius r. Then, as we continue this random walk, when the distance between x_k and the defined boundary is smaller than some ϵ, the walk is terminated and the value of the closest boundary value stored. By taking many walks, the expected value of this process is equal to the value of $u(x_0)$. Additional algorithms include the Walk on Stars method, Walk on Boundary method, and Monte Carlo Finite Element Method [18] [20] [8]. These all provide viable avenues for developing faster PDE solvers for computer graphics.

2.2. Geometric Applications

The past few years has seen an increasing need to the development of faster geometric processing algorithms. The WoS algorithm was recently applied to geometry processing tasks for elliptic PDEs [16], that was soon adapted to work with Neumann and Dirichlet boundary conditions, and other applications [17]. These algorithms could solve PDEs on various solid regions in 2D and 3D, allowing for flexibility with various geometric shapes and avoiding comment challenges such as mesh generation with conventional methods [16]. However, many of these methods still operate only on a subset of PDEs with high variance and expensive computation.

2.3. Neural Networks

Previous work has explored the use of Physics-informed neural networks to approximate the solution to a PDE. PINNs work by utilizing a physics-informed self-supervised loss function with the given forcing function and boundary conditions to generate accurate solutions. These models have been applied to solve material systems and power systems [14] [15]. It has also been proposed that combining Monte Carlo PDE solvers using WoS with information retrieval techniques may allow for faster computational times, more robust models, and lower variance [16]. We draw inspiration from previous models utilizing neural fields to cache previous results from Monte Carlo solvers to generate solutions [10].

Specifically, Li et al. [2023] developed a neural caching solution that employed a hybrid Monte Carlo PDE solver with a neural field. By training the neural field to approximate the solution to a PDE using the Monte Carlo WoS PDE solver, the neural field was then used during test time to reduce variance in the actual Monte Carlo PDE solver by querying from the neural field after a certain number of steps.

3. Dataset

To generate our dataset, we generated a random set of 1000 elliptic PDEs with Dirichlet boundary conditions. This is done through generating a random forcing and boundary function for each equation, with each function being a random composition of up to 5 exponential, polynomial or trigonometric functions, dictating the behavior of the PDE. We used a 1 x 1 square as the boundary shape for these PDEs.

Figures 1 and 2 show a random sample visualization of these PDEs.

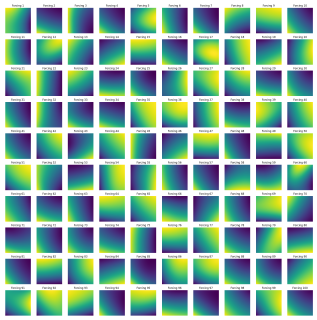

Figure 1: Dataset of PDEs Run on Baseline Neural Network Model

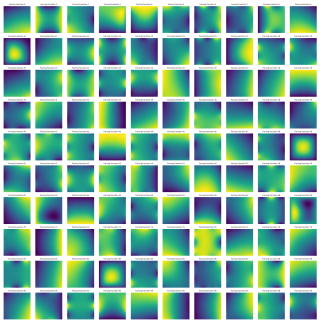

Figure 2: Dataset of PDEs Run on WoS Solver for 10 Iterations

Our dataset was split into 80-20 train and test dataset, with the same test dataset only used for evaluating the final evaluations for all methods.

4. Methods

4.1. Monte Carlo PDE Solvers

In this paper, the elliptic PDEs that we are focusing on are in the following form, as referenced from [10]

For $x \in \Omega$:

$$\nabla \cdot (\alpha(x)\nabla u(x)) + \omega(x)\nabla u(x) - \sigma(x)u(x) = -f(x)$$

For $x \in \partial\Omega$

$$u(x) = g(x)$$

where $\alpha, \omega,$ and σ are spatially varying coefficients, f is the forcing function, g is our boundary function, and Ω is defined by the signed distance function., In our model, we focus on solving 2D PDEs.

Monte Carlo PDE solvers expresses the numerical solution of a PDE using an integral equation. The solution to this type of PDE is as follows

$$u(x) = S(x) + \int_{B_r(x)} u(y)G_x(y)dy + \int_{\partial B_r(x)} u(z)K_X(z)dz$$

where one integral is evaluated over a ball centered at x with radius r and the other is evaluated over the boundary of the ball. The other terms $S(x)$, $G_X(y)$, and $K_X(z)$ are dependent on Green's function and the Poisson kernel. More detailed definitions are provided in [10] and [16]. The Monte Carlo PDE solver method then steps through each point x within the domain space and numerically runs a certain number of iterations at each x to approximate the solution $u(x)$.

4.2. Physics-informed Neural Networks

As universal function approximators, neural networks have been shown to be useful in approximating the solution to a PDE u(x). In particular, PINNs have been shown to acheive very good albiet biased results [7]. In our paper, we implemented a PINN for the poisson equation in a domain Ω with Dirichlet boundary conditions can be written as:

$$\begin{cases} \Delta u(x) = f(x) & \text{for } x \in \Omega, \\ u(x) = g(x) & \text{for } x \in \partial\Omega, \end{cases} \tag{1}$$

With this formulation, we can approximate the solution u(x) with a PINN $\hat{u}(x)$ trained on the loss function which combines the residual of the Poisson equation in the domain and the boundary conditions. The total loss $L(\theta)$ can be expressed as:

$$L(\theta) = L_{\text{PDE}}(\theta) + L_{\text{BC}}(\theta), \tag{2}$$

where

$$L_{\text{PDE}}(\theta) = \frac{1}{N_\Omega} \sum_{i=1}^{N_\Omega} \left|\Delta\hat{u}(x_i;\theta) - f(x_i)\right|^2, \tag{3}$$

and

$$L_{\text{BC}}(\theta) = \frac{1}{N_{\partial\Omega}} \sum_{i=1}^{N_{\partial\Omega}} \left|\hat{u}(x_i;\theta) - g(x_i)\right|^2. \tag{4}$$

Here, N_Ω and $N_{\partial\Omega}$ are the number of collocation points in the domain and on the boundary, respectively. The optimization objective is then shown to be $\theta^* = \arg\min_\theta L(\theta)$.

Through this method, neural networks can be trained to approximate the solution to vairous types of PDEs with initial and boundary conditions. In our project, we employ a deep learning library DeepXDE recently developed to allow for efficient development of PINNs [11].

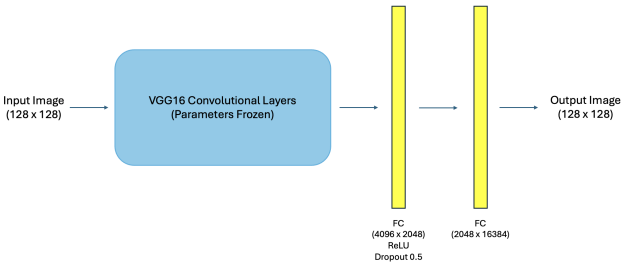

Figure 3: VGG-16 Model Diagram

4.3. Fine-tuning VGG-16

While much of previous literature focuses on utilizing multi-layer perceptrons (MLP) with sinusoidal activations or neural fields to achieve desired results within computational limits [10] , we decided to explore the possibilities of utilizing convolutional neural networks to make predictions of the ground truth solution given an early Monte Carlo iteration. We first utilized the VGG-16 model [19] pre-trained on the ImageNet dataset provided by PyTorch. To recast the properties of the VGG-16 model to suit our needs of image generation, we fine-tuned the fully connected layers of the model. The model was loaded using models.vgg16 and the parameters of the convolutional layer and first two fully connect layers were frozen. The last two linear layers were unfrozen and trained using 80 percent of the data that was generated as part of the dataset. The second to the last layer consisted of a linear layer of size 4096×2048 followed by a ReLu and dropout layer. The last linear layer was of size $2048 \times (128 * 128)$ since the image sizes were 128×128. The model is shown in Figure 3.

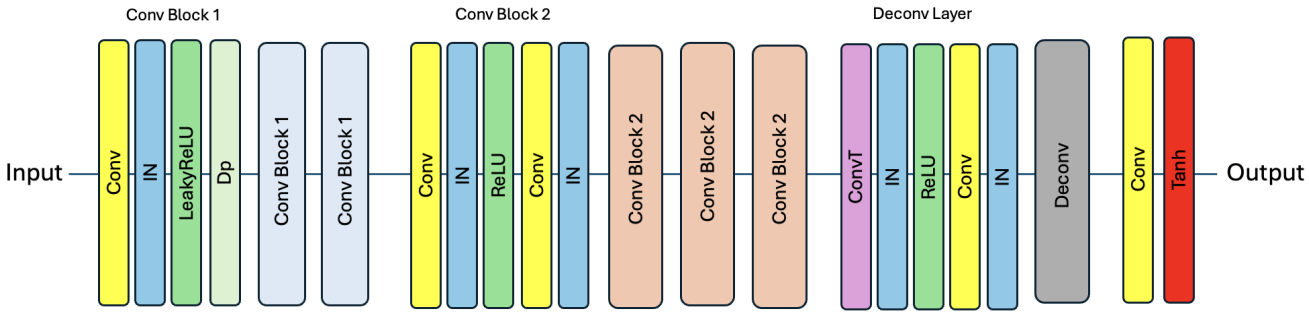

Figure 4: Generator Network Architecture

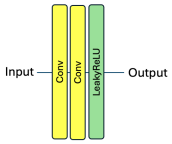

Figure 5: Discriminator Network Architecture

For our model, we utilized the mean square error (MSE) loss function to train our model. We used this because this is how our baseline errors was measured and provided a suitable loss function to penalize larger pixel errors. We then experimented with various optimizers including Adam, AdamW, and RMSProp. These were individually trained and the metric were compared against each other.

4.4. Generative Adversarial Networks

To further explore possibilities of image generation, we sought to explore the use of generative adversarial networks, recently popularized to generate "fake" images from input. GANs operate by training two neural networks, a generator and a discriminator, to compete with each other. The generate seeks to generate more and more realistic images, while the discriminator seeks to discriminate between the generated image and the ground truth.

To explore possibilities of implementing GANs, we adopted the methodology used on Github [1]. The GAN was then appropriately modified to take in inputs of the shape 128 x 128 and output results of the shape 128 x 128. The generator primarily contains a few convolutional layers, followed another convolutional block, and finally followed by deconvolutional layers. The residual blocks initially present in the developed GAN were removed to reduce the complexity of the overall structure of the GAN and stabilize loss during training. The discriminator was additionally modified to only consist of two convolutional layers. The model structure is shown in Figures 4 and 5.

4.4.1 Model Evaluation

To evaluate our trained models, we utilized 20 percent of our generated dataset to test the behavior of our model. We measured the metrics of mean squared error (MSE), root mean squared error (RMSE), signal-to-noise ratio (SNR) and peak signal-to-noise ratio (PSNR). To measure time taken for the various PDE solvers, we evaluated them on Google Colab, running a T4 GPU with 16Gb of GPU RAM and 16Gb of CPU RAM.

4.5. SUNet

SUNet, or Swin-UNet, is a neural network architecture designed for image restoration tasks [4]. It begins with a 3x3 convolutional layer to generate shallow feature maps, followed by 5 layers of Swin Transformer Blocks, Patch Merging, and Dual Upsampling. The Swin Transformer employs a hierarchical structure and a shifted window mechanism to capture both local and global contextual information efficiently. This combination with the UNet architecture enables effective feature extraction and reconstruction of noisy images. In other words, while VGG-16 is typically used for image classification, SUNet was designed for image restoration. Leveraging this, we apply a SUNet pre-trained on denoising tasks in order to "denoise" the 50th iteration Monte Carlo solution.

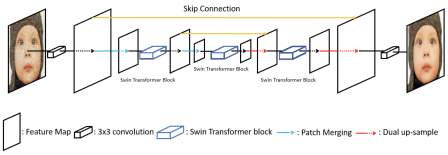

Figure 6: SUNet Architecture from [4]

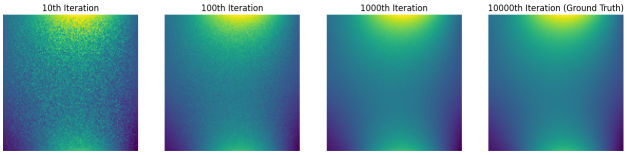

Figure 7: Example Solution of WoS Baseline Solver

5. Baseline Results

5.1. WoS Baseline

For our Monte Carlo PDE solver baseline, we utilized the Walk-on-Spheres method implementation based on Li et al. [2023] and Sawhney and Crane [2020] to generate solutions at various interactions using the dataset of forcing functions and boundary conditions. The signed distance function used was in the shape of a 1 by 1 square centered at $(0.5, 0.5)$. We first used PDE equations with known analytical solutions and compared it with WoS results at 10000th iteration. Noticing that MSE $< 10^{-5}$, we used 10000th iteration results as ground truth for general PDEs. An example of a set of results generated by the WoS baseline solution is shown in Figure 6, displaying the Monte Carlo generated solutions after 10 iterations, 100 iterations, 1000 iterations, and 10,000 iterations. To quantify the accuracy of our results, we calculated the element-wise mean squared error (MSE), root mean squared error (RMSE), signal-to-noise ratio (SNR), and peak signal-to-noise-ratio (PSNR) for each forcing function. These results are shown in Table 1. We also analyzed the behavior of these quantities as a function of the number of iterations that the Monte Carlo PDE solver runs. These are shown in Figure 7.

Metric	10 Iterations	100 Iterations	1000 Iterations
MSE	0.01084	0.00117	0.00010
RMSE	0.08849	0.02818	0.00883
SNR	16.76034	26.77800	36.58800
PSNR	21.98031	31.99797	41.80797

Table 1: Metrics for Walk-On-Spheres (WoS)

5.2. PINN Baseline Model

For our neural network baseline model, we utilized a pretrained Physics-informed neural network (PINN) to solve the same dataset of forcing functions and boundary conditions. The PINNs were trained for 10000 steps using the DeepXDE model and the results of the model are shown in Table 2.

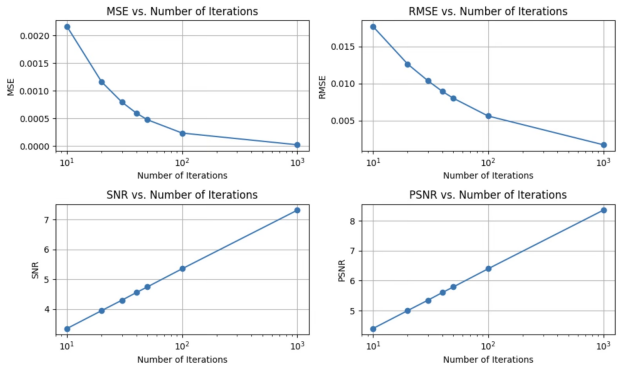

Figure 8: Metrics for WoS Baseline Solver

MSE	RMSE	SNR	PSNR
0.31441	0.54956	−0.16089	2.17732

Table 2: Metrics for PINN

6. Experiments

In the following section, we detailed the specific training procedures utilized to train each of our models and the results that we obtained.

6.1. VGG-16

The model was initially trained using the Adam optimizer with a mean squared error loss function. The learning rate was intitially set at 0.001. After sampling various learning rates, we found that using 0.0001 was optimal for loss convergence. The batch sized used was 64, although this could be experimented with in the future to better fine-tune the hyperparameters of our model. To further explore potential ways to optimize this model, we explored other optimizers such as AdamW and RMSProp. Experimenting with the three different optimizers, we trained each optimizer for 200 epochs. The resulting loss function is shown in Figure 8. We observe a much faster initial decrease using the Adam optimizer, but all three losses eventually end up converging to around the same value.

After running this fine-tuned model on our dataset for 200 epochs, we achieved these final results. Unfortunately, the final results we achieved using a fine-tuned VGG-16 were suboptimal and performed worse than the input image. Due to the fact that VGG-16 is primarily used as an image classifier, it may not be the best candidate for image denoising and generation tasks. However, we do show that possibly with further fine-tuning, it is possible for VGG-16 to be implemented for denoising tasks, as shown in [12] and [5].

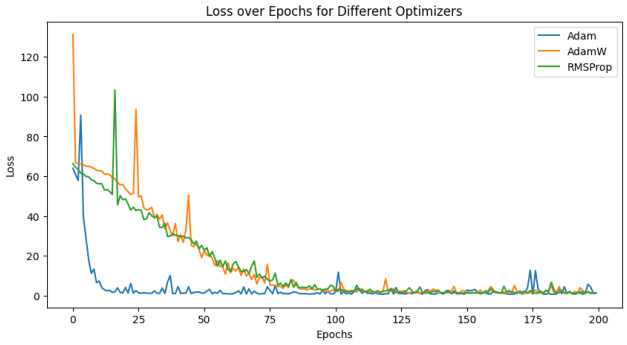

Figure 9: Loss Function of Different Optimizers

MSE	RMSE	SNR	PSNR	Inference Time
0.0263	0.1384	13.5362	18.7537	0.00052 s

Table 3: Metrics for Fine-Tuned VGG-16

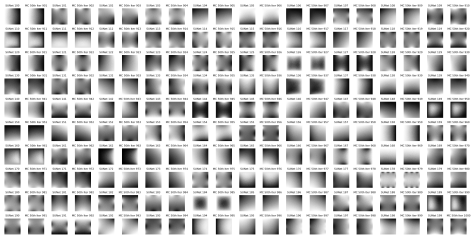

Figure 10: SUNet Image Denoising Results

6.2. SUNet

For implementation, modifications were made to adapt SUNet for grayscale inputs by converting single-channel images to three channels. We then (a) directly applied inference without fine tuning and (b) fine-tuned on 1000 50th-iteration monte-carlo solutions and their corresponding 1000th-iteration solution, with an 80-20 split for training and validation. We found, however, that the fine-tuned model had a tendency to "blur" the whole image such that the variance of the pixel values were so low as to eliminate detail almost entirely. Therefore, we present the results with pure inference.

MSE	RMSE	SNR	PSNR	Inference Time
0.00722	0.05964	21.31592	26.53588	0.157 s

Table 4: Metrics for SUNet

6.3. GAN

To train the GAN, the hyperparameters were tuned to optimize the rate at which the generator and discriminator

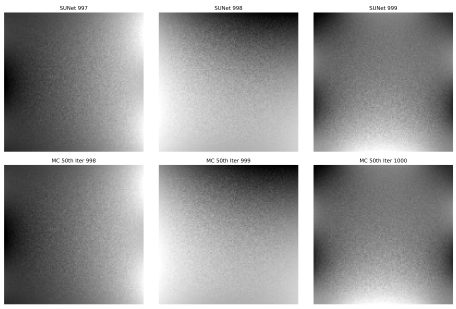

Figure 11: SUNet Image Denoising Results, 3 Data Points

loss decreased. Initially, the generator loss was decreasing at a much slower rate than the discriminator loss, causing the generator's parameters to stop updating and for the loss to stall very early on. To fix this and improve model performance, a few updates were made. The learning rate on the optimizer for the generator was increased to be 0.0006, while the optimizer for the discriminator used a learning rate of 0.00002. Furthermore, a cosine annealing learning rate scheduler was used for the generator to improve loss performance. We used 200 epochs to train the GAN, although potentially more could be used to achieve even higher performance in the future. The loss function used in this model as a mean squared error loss with an LSGAN model. The generator and discriminator loss over epochs is presented in Figure 10.

Metric	Average	Standard Deviation
Generator Loss	0.2898	0.0101
Discriminator Loss	0.2565	0.0035
MSE	0.0015	0.0005
RMSE	0.0332	0.0092
SNR	25.6638	2.3795
PSNR	29.9596	2.2977
Inference Time	0.000716 s	0.000121 s

Table 5: GAN Results over 10 Training Sessions with Random Initialization

For a more detailed analysis of the results, we present some of the sample images that were generated by the GAN in comparison to the input image (10th iteration of MC PDE solver) and the ground truth in Figure 11 on the next page. We see from observation that the GAN provides a smoother and more accurate solution compared to the 10th iteration.

6.4. Discussion

We tried out several hybrid WoS and neural network methods, and found that our hybrid WoS-GAN method performs very well. Inference times are much lower than the

Method	MC Iterations	MSE	RMSE	SNR	PSNR
WoS	1000	0.00010	0.00883	36.58800	41.80797
WoS	10	0.01084	0.08849	16.76034	21.98031
WoS	100	0.00117	0.02818	26.77800	31.99797
PINN	N/A	1.51897	0.91412	−0.16089	2.17732
Hybrid WoS-VGG	10	0.0263	0.1384	13.5362	18.7537
Hybrid WoS-SUNet	10	0.00722	0.05964	21.31592	26.53588
Hybrid WoS-GAN	10	**0.0015**	**0.0332**	**25.6638**	**29.9596**

Table 6: Comparison of Model Performance Metrics

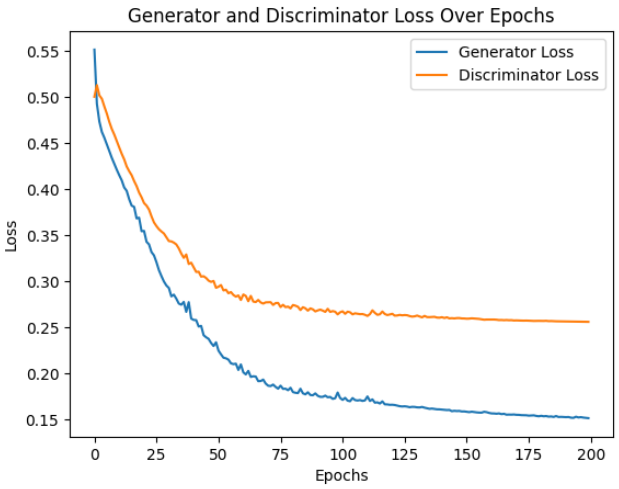

Figure 12: GAN Generator and Discriminator Loss

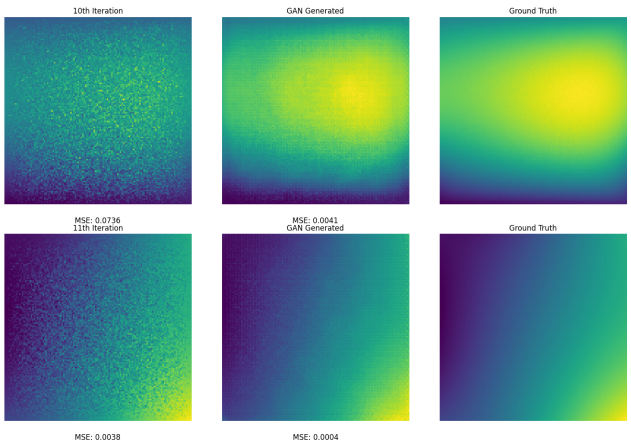

Figure 13: GAN Generated Images (Random Sample of 2 PDEs)

other neural networks that we have tried. In particular, our hybrid WoS-GAN solver was able to achieve similar performance to the 100th iteration of WoS despite taking in only the 10th iteration as an input. With an inference time that is negligible (0.000716s per PDE) compared to the time taken for WoS (1.211s per PDE for 100 iterations), we are able to achieve an almost tenfold increase in speed for the same performance. With a more complex GAN model, a larger training dataset, it is possible to achieve even better results using a GAN for image denoising.

Notably, during experimentation and testing a wide variety of applicable neural network models as well as image denoising model, a common characteristic was the need for three channel images. This means that we had to reshape and extrapolate our WoS solutions to fit this criteria, resulting in us converting back and forth between image dimensions. This likely degrades the perfomrance of the models that we trained. However, specifically for our GAN, we kept our images in grayscale, which may have been beneficial for its better performance.

Additionally, with more time, it would have definitely been beneficial to train and tune our models further. During training, using the T4 GPU from Google Colab, we observed relatively fast training times for the GAN and VGG-16 models, meaning that there is potential to expand to larger datasets with more training epochs to further boost our model performance.

7. Conclusion

Herein, we have demonstrated how a hybrid method of Monte Carlo and Neural networks allows us to solve PDEs in a much faster manner as compared to current Monte Carlo methods, while greatly increasing accuracy as compared to PINNs, the current state-of-the-art for neural network PDE solvers. In particular, our hybrid WoS-GAN solver was able to achieve similar performance to 100 iterations of WoS, despite only running WoS for 10 iterations, and thus giving an almost tenfold increase in speed.

A key limitation of our study, however, is that we limited our dataset to 2D PDEs, with a fixed square boundary, which limits the current applicability of this study. A good area for future work would thus be extending the result

found here to a 3D space, and possible methods for doing so, including either slicing it into 2D slices or considering a hybrid WoS-3D Convolutional Neural Network design.

We also limited ourselves to elliptic PDEs and Dirichlet boundary conditions, and it will be useful to see if the same property holds for other boundary conditions, including Newmann boundaries which are also common in graphics applications.

Another potential area for future research is to further develop the SUNet model for image denoising tasks and fine-tuning it to denoise earlier iterations of the WoS PDE solver. Additionally, during model exploration, we noticed that many models require RGB images to train and test, meaning that the inputs must be 3 channels. More research into the development of more robust models to handle grayscale images may be beneficial toward faster PDE solvers and help computer graphics applications.

Nonetheless, we have shown conclusively that a hybrid Monte Carlo and Neural Network approach to PDE solving can reap both the benefits of unbiasedness and accuracy from Monte Carlo solutions, and efficiency from Neural Network solutions, and believe that this work would serve as a good foundation for future PDE solvers.

8. Contributions

- **Hong Meng Yam**: Reviewed literature, helped implement WoS solver, generated datasets, worked on various methods including CNNs, MAXIM, and VGG. Worked on milestone and final report.

- **Ethan Hsu**: Reviewed literature, implemented PINN, implemented and fine tuned SUNnet, worked on other image restoration neuronetworks such as MAXIM and CGNet, helped draft and implement final report.

- **Ivan Ge**: Reviewed relevant literature and pre-existing models, helped build Monte Carlo WoS solver, worked on coding VGG fine-tuning model and GAN approach. Benchmarked models for relevant characteristics to compare to baseline methods. Helped draft and edit final report.

9. Acknowledgements

We received help from Guandao Yang, and met with him for advice twice for this project. We also received help from Bohan Wu, CS231N TA, and met with him for advice once for this project.

References

[1] R. R. Abeer Alsaiari, Manu Mathew Thomas. Image denoising using a generative adversarial network. 4

[2] J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018. 1

[3] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022. 1

[4] C.-M. Fan, T.-J. Liu, and K.-H. Liu. Sunet: Swin transformer unet for image denoising. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2022. 4

[5] M. Hami and M. JameBozorg. Assessing the impact of cnn auto encoder-based image denoising on image classification tasks, 2024. 5

[6] M. Horie and N. MITSUME. Physics-embedded neural networks: Graph neural pde solvers with mixed boundary conditions. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 23218–23229. Curran Associates, Inc., 2022. 1

[7] Z. Jiang, J. Jiang, Q. Yao, and G. Yang. A neural network-based pde solving algorithm with high precision. *Scientific Reports*, 13(1):4479, 2023. 1, 3

[8] C. L. D. J. Jin, Guoliang. Monte carlo finite element method of structure reliability analysis. *Reliability Engineering System Safety*, 40(1), 1993. 2

[9] Y. Khoo, J. Lu, and L. Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021. 1

[10] Z. Li, G. Yang, X. Deng, C. De Sa, B. Hariharan, and S. Marschner. Neural caches for monte carlo partial differential equation solvers. In *SIGGRAPH Asia 2023 Conference Papers*, SA '23, New York, NY, USA, 2023. Association for Computing Machinery. 1, 2, 3

[11] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *CoRR*, abs/1907.04502, 2019. 3

[12] M. Mehdizadeh, C. MacNish, D. Xiao, D. Alonso-Caneiro, J. Kugelman, and M. Bennamoun. Deep feature loss to denoise oct images using deep neural networks. *Journal of Biomedical Optics*, 26(4):046003, 2021. 5

[13] M. Muller. Some continuous monte carlo methods for the dirichlet problem. *Ann. Math. Statist.*, 27(3), 1956. 2

[14] S. A. Niaki, E. Haghighat, X. Li, T. Campbell, and R. Vaziri. Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. *CoRR*, abs/2011.13511, 2020. 2

[15] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017. 2

[16] R. Sawhney and K. Crane. Monte carlo geometry processing: A grid-free approach to pde-based methods on volumetric domains. *ACM Trans. Graph.*, 39(4), 2020. 1, 2, 3

[17] R. Sawhney, D. Seyb, W. Jarosz, and K. Crane. Grid-free monte carlo for pdes with spatially varying coefficients. *ACM Trans. Graph.*, 41(4), jul 2022. 2

[18] R. Sawhney, D. Seyb, W. Jarosz, and K. Crane. Walk on stars: A grid-free monte carlo method for pdes with neumann boundary conditions. *ACM Trans. Graph.*, 2023. 1, 2

[19] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. 3

[20] R. Sugimoto, T. Chen, Y. Jiang, C. Batty, and T. Hachisuka. A practical walk-on-boundary method for boundary value problems. *ACM Transactions on Graphics*, 42(4):1–16, July 2023. 2

[21] R. Zhang, Q. Meng, R. Zhu, Y. Wang, W. Shi, S. Zhang, Z.-M. Ma, and T.-Y. Liu. Monte carlo neural pde solver for learning pdes via probabilistic representation, 2023. 1