Learning Solution Operators for Partial Differential Equations via Monte Carlo-Type Approximation

Salah Eddine Choutri
NYUAD Research Institute
New York University Abu Dhabi
United Arab Emirates, UAE
sc8101@nyu.edu

Prajwal Chauhan
Engineering Division
New York University Abu Dhabi
United Arab Emirates, UAE
pc3377@nyu.edu

Othmane Mazhar
Laboratoire de Probabilités, Statistique et Modélisation
Sorbonne University & Université Paris Cité
Paris, France
omazhar@lpsm.paris

Saif Eddin Jabari
Engineering Division
New York University Abu Dhabi
United Arab Emirates, UAE
sej7@nyu.edu

Abstract

The Monte Carlo-type Neural Operator (MCNO) introduces a lightweight architecture for learning solution operators for parametric PDEs by directly approximating the kernel integral using a Monte Carlo approach. Unlike Fourier Neural Operators, MCNO makes no spectral or translation-invariance assumptions. The kernel is represented as a learnable tensor over a fixed set of randomly sampled points. This design enables generalization across multiple grid resolutions without relying on fixed global basis functions or repeated sampling during training. Experiments on standard 1D PDE benchmarks show that MCNO achieves competitive accuracy with low computational cost, providing a simple and practical alternative to spectral and graph-based neural operators.

1 Introduction

Neural operators extend the success of neural networks from finite-dimensional vector mappings to infinite-dimensional function spaces, enabling the approximation of solution operators for partial differential equations (PDEs). These models learn mappings from problem inputs, such as coefficients or boundary conditions, to PDE solutions, allowing rapid inference and generalization across parameterized problem families. Early architectures like DeepONet [6] use branch and trunk networks, while Fourier Neural Operators (FNO) [5] leverage spectral convolutions and FFTs for efficient operator approximation on uniform grids. Wavelet-based models such as MWT [1] and WNO [7] improve spatial localization, and graph-based approaches like GNO [4] extend operator learning to irregular domains.

We propose the Monte Carlo-type Neural Operator (MCNO), a lightweight architecture that approximates the integral kernel of a PDE solution operator via Monte Carlo sampling, requiring only a single random sample at the beginning of training. By avoiding spectral transforms and deep hierarchical architectures, MCNO offers a simple yet efficient alternative that balances accuracy and computational cost. We evaluate MCNO on standard 1D PDEs, including Burgers' and Korteweg–de Vries equations, using benchmark datasets, and show competitive performance against existing neural operators while maintaining architectural simplicity. Our contributions include the MCNO design,

Machine Learning and the Physical Sciences Workshop, NeurIPS 2025.

feature mixing for spatial and cross-channel dependencies, an interpolation mechanism for structured grids and benchmark evaluation.

2 Monte Carlo-type Neural Operator (MCNO)

Neural operators provide a framework for learning mappings between infinite-dimensional function spaces, enabling efficient approximation of solution operators for parametric PDEs [2]. For a generic PDE, the true solution operator $G^\dagger$ is a mapping that takes an input a to the corresponding PDE solution u s.t $G^\dagger : \ a \mapsto u = G^\dagger(a)$. For many inputs, evaluating $G^\dagger$ with classical solvers can be prohibitively expensive. Given a dataset $\{(a_j, u_j)\}_{j=1}^N$ with $u_j = G^\dagger(a_j)$, the goal is to learn a parameterized approximation $G_\theta \approx G^\dagger$ by minimizing a suitable loss. A neural operator adopts iterative architectures to compute such approximate operator G_θ, to this end it lifts the input to a higher-dimensional representation $v_0(x) = P\big(a(x)\big), \ x \in \mathcal{D} \subset \mathbb{R}^d$ via a local map P. It then, updates it through a linear transformation via matrix W and a kernel-integral transformation $(\mathcal{K}_\phi v)(x)$ with nonlinear activations σ. The $(t+1)$th, $t \in \mathbb{N}$, neural operator layer is expressed as:

$$v_{t+1}(x) = \sigma\left(W v_t(x) + (\mathcal{K}_\phi v_t)(x)\right), \quad x \in \mathcal{D},$$

with

$$(\mathcal{K}_\phi v_t)(x) = \int_D \kappa_\phi(x, y, a(x), a(y)) v_t(y) dy,$$

where κ_ϕ is a learned kernel inspired by the Green function commonly involved in the solution of linear PDEs. A final projection Q maps the final features back to the target function space

$$\big(G_\theta(a)\big)(x) = Q\big(v_T(x)\big), \qquad x \in \mathcal{D}.$$

MCNO follows this framework but replaces the integral with a Monte Carlo estimate over a fixed set of sampled points, i.e.,

$$(\hat{\mathcal{K}}_{N,\phi} v_t)(x) = \frac{1}{N} \sum_{i=1}^{N} \kappa_\phi(x, v_t(y_i)) v_t(y_i). \tag{1}$$

This avoids spectral transforms or hierarchical architectures and achieves linear complexity in the number of sampled points. To enhance efficiency, MCNO leverages PyTorch's `einsum` to aggregate features across spatial locations and channels jointly by expressing the kernel $\kappa_\phi(x, v_t(y_i))$ as

$$\kappa_\phi(x, v_t(y_i)) = \sum_j \phi_j \, v_{t,j}(y_i),$$

allowing GPU parallelization and avoiding costly matrix multiplications. Since the Monte Carlo integral is computed on a subset of points, results are interpolated to reconstruct a structured grid representation compatible with standard neural architectures. The interpolation step maps sparse Monte Carlo estimates onto a dense discretized domain while preserving the local function structure. We employ continuous interpolation, typically using linear interpolation for smooth approximations while maintaining efficiency.

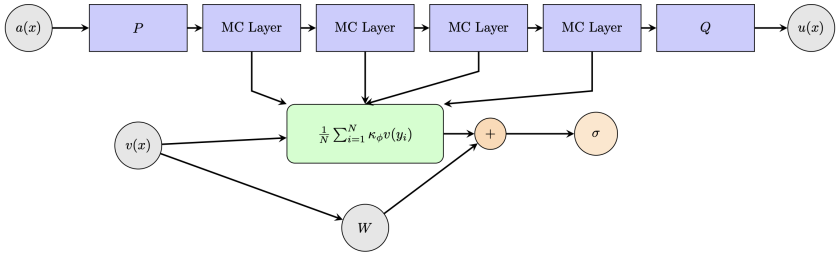

Figure 1: MCNO architecture.

Procedure 1 MCNO — Forward Pass

1: **Given:** parameters ϕ, W; samples $\{y_i\}_{i=1}^N$; grid; activation σ; interpolation rule
2: **Input:** current representation v_t
3: **Output:** $v_{t+1}(x)$
4: Estimate kernel: $(\hat{\mathcal{K}}_{N,\phi} v_t)(x) = \frac{1}{N}\sum_{i=1}^N \kappa_\phi\big(x, v_t(y_i)\big)\, v_t(y_i)$
5: Interpolate $(\hat{\mathcal{K}}_{N,\phi} v_t)$ onto the full grid
6: Update: $v_{t+1}(x) = \sigma\Big(W v_t(x) + (\hat{\mathcal{K}}_{N,\phi} v_t)(x)\Big)$

3 Bias–Variance Analysis and Dimensional Scaling

At each layer, MCNO computes the kernel aggregation (1) using N points $\{y_{s_i}\}_{i=1}^N$ sampled uniformly from the grid $\{y_j\}_{j=1}^{N_{\text{grid}}} \subset D \subset \mathbb{R}^d$. Each sample–feature update is used once, so the per-layer cost is $\text{Cost}_{\text{MCNO}} = \mathcal{O}(N\, d_v)$ (feature width d_v). In practice, tensor contractions parallelized over x give aggregation $\mathcal{O}(N)$, while interpolation/reconstruction scales with N_{grid}.

Error decomposition. Define

$$E(x) = (\hat{\mathcal{K}}_{N,\phi} v_t)(x) - (\mathcal{K}_\phi v_t)(x).$$

Adding and subtracting $(\mathcal{K}_\phi^{\text{grid}} v_t)(x) = \frac{1}{N_{\text{grid}}}\sum_{j=1}^{N_{\text{grid}}} \kappa_\phi(x, y_j) v_t(y_j)$ yields

$$E(x) = \underbrace{(\mathcal{K}_\phi^{\text{grid}} v_t)(x) - (\mathcal{K}_\phi v_t)(x)}_{\text{Bias}} + \underbrace{(\hat{\mathcal{K}}_{N,\phi} v_t)(x) - (\mathcal{K}_\phi^{\text{grid}} v_t)(x)}_{\text{Variance}}.$$

Bias. Partition D into hypercubes $\{C_j\}_{j=1}^{N_{\text{grid}}}$ with side h so $N_{\text{grid}} = (1/h)^d$. For each C_j centered at y_j, $\int_{C_j} \kappa_\phi(x,y) v_t(y)\, dy = \kappa_\phi(x, y_j) v_t(y_j) h^d + R_j(x)$ with $|R_j(x)| \le L\, \frac{\sqrt{d}}{2}\, h^{d+1}$, hence

$$\big|(\mathcal{K}_\phi^{\text{grid}} v_t)(x) - (\mathcal{K}_\phi v_t)(x)\big| \le \sum_{j=1}^{N_{\text{grid}}} |R_j(x)| \le L\, \frac{\sqrt{d}}{2}\, h\, N_{\text{grid}} h^d = C_1 N_{\text{grid}}^{-1/d}. \tag{2}$$

Variance. For fixed x, $(\hat{\mathcal{K}}_{N,\phi} v_t)(x) = \frac{1}{N}\sum_{i=1}^N Z_i(x)$ with $Z_i(x) = \kappa_\phi(x, y_{s_i}) v_t(y_{s_i})$, $|Z_i(x)| \le C$. Hoeffding gives

$$\mathbb{P}\Big(\big|(\hat{\mathcal{K}}_{N,\phi} v_t)(x) - \mathbb{E}[(\hat{\mathcal{K}}_{N,\phi} v_t)(x)]\big| > t\Big) \le 2\exp\Big(-\frac{Nt^2}{2C^2}\Big).$$

By a union bound over $\{x_j\}_{j=1}^{N_{\text{grid}}}$ and failure δ,

$$\sup_{x\in D} \big|(\hat{\mathcal{K}}_{N,\phi} v_t)(x) - (\mathcal{K}_\phi^{\text{grid}} v_t)(x)\big| \;\le\; C_2 \sqrt{\frac{\log(2N_{\text{grid}}/\delta)}{N}}. \tag{3}$$

Sample complexity and cost. From (2)–(3),

$$\sup_{x\in D} |E(x)| \;\lesssim\; C_1 N_{\text{grid}}^{-1/d} + C_2 \sqrt{\frac{\log N_{\text{grid}}}{N}}.$$

To reach tolerance ε, choose $N_{\text{grid}} = \mathcal{O}(\varepsilon^{-d})$ and $N = \mathcal{O}(\varepsilon^{-2}\log(\varepsilon^{-d}))$. Since aggregation is $\mathcal{O}(N)$ and reconstruction is $\mathcal{O}(N_{\text{grid}})$, the total cost is

$$\text{Cost}_{\text{MCNO}}(\varepsilon, d) = \mathcal{O}(N) + \mathcal{O}(N_{\text{grid}}) = \tilde{\mathcal{O}}(\varepsilon^{-2} + \varepsilon^{-d}),$$

with $\tilde{\mathcal{O}}$ hiding logarithms.

High-dimensional note. Dimension d enters via the bias (through $N_{\text{grid}}^{-1/d}$) and grid cost; the Monte Carlo variance, and thus the stochastic sample complexity, remains dimension-independent.

4 Numerical experiments

We evaluate the Monte Carlo-type Neural Operator on two standard benchmark problems: Burgers' equation and Korteweg-de Vries (KdV) equation. These benchmarks are widely used for testing operator learning methods due to their diversity in complexity and relevance to applications. To ensure fairness and consistency, we adopt the same datasets and experimental setups as described in [5]. The MCNO employs a four-layer architecture of Monte Carlo kernel integral operators, with ReLU activations. Training uses 1000 samples, with 100 additional samples reserved for testing. Optimization is performed with the Adam optimizer, starting at a learning rate of 0.001, halved every 100 epochs over a total of 500 epochs. We set the width to 64, $N = 100$ for Burgers' equation and $N = 75$ for Korteweg-de Vries (KdV) equation. All experiments are conducted on an Tesla V100-SXM2-32GB of memory GPU and the accuracy is reported in terms of relative L_2 loss. The errors for the main benchmarks FNO, MWT and WNO are reported with our implementation on the same GPU that we used to train our model. For the other benchmarks: GNO, LNO and MGNO, the errors are taken from [1]. These models were trained and tested on an Nvidia V100-32GB GPU, therefore the comparison with their reported errors is fair to a certain extent.

4.1 Burgers' Equation: Dataset and Results

We consider the one-dimensional Burgers' equation, a nonlinear PDE commonly used to model viscous fluid flow, with periodic boundary conditions and initial condition $u(x, 0) = u_0(x)$, where $u_0 \in L^2_{\text{per}}((0, 1); \mathbb{R})$ and $\nu > 0$ is the viscosity: $\partial_t u + \partial_x(u^2/2) = \nu \partial_{xx} u$, $x \in (0, 1)$, $t \in (0, 1]$. Following the setup of the Fourier Neural Operator (FNO) benchmark [5], the initial condition is sampled from a Gaussian $\mu = \mathcal{N}(0, 625(-\Delta + 25I)^{-2})$, and the PDE is solved with a split-step method on a high-resolution grid (8192 points) and subsampled for training and testing. The task is to learn the operator mapping $G^\dagger : u_0 \mapsto u(\cdot, 1)$. Table 1 summarizes results of different neural operators across resolutions: the proposed Monte Carlo-type Neural Operator (MCNO) achieves competitive performance, outperforming GNO, LNO, MGNO, and FNO in speed and accuracy. While WNO performs best on higher resolutions, it is inconsistent across grids and slower than MCNO and FNO. The MWT Leg model achieves slightly lower consistent errors but is computationally expensive. Overall, MCNO offers a balanced trade-off between accuracy, efficiency, and adaptability, with relative L_2 error and per-epoch time remaining stable as the number of samples increases (Figures 2 and 3).

Table 1: Benchmarks on 1-D Burgers' Equation showing relative L_2 errors for different input resolutions s.

Model	Time per epoch (s=256, s=8192)	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
GNO [4]	-	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO [6]	-	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO [3]	-	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
WNO [7]	(2.45s, 3.1s)	0.0546	0.0213	0.0077	0.0043	0.0027	0.0012
FNO [5]	(0.44s, 1.56s)	0.0183	0.0182	0.0180	0.0177	0.0172	0.0168
MCNO	(0.40s, 1.32s)	0.0064	0.0067	0.0062	0.0069	0.0071	0.0065
MWT Leg [1]	(3.70s, 7.10s)	0.0027	0.0026	0.0023	0.0024	0.0025	0.0023

4.2 Korteweg-de Vries (KdV) equation: Dataset and Results

We consider the one-dimensional Korteweg-de Vries (KdV) equation, a nonlinear PDE modeling shallow water waves and dispersive phenomena, defined as $\partial_t u = -0.5\, u \partial_x u - \partial_x^3 u$, $x \in (0, 1)$, $t \in (0, 1]$. Following the dataset setup in [1], the initial conditions $u_0(x) = u(x, 0)$ are sampled from Gaussian random fields with periodic boundary conditions, $u_0 \sim \mathcal{N}(0, 7^4(-\Delta + 7^2 I)^{-2.5})$, and high-resolution solutions are computed with Chebfun (2^{10} points) and subsampled for training and testing. The task is to learn the operator mapping $u_0 \mapsto u(\cdot, 1)$. Table 2 reports benchmark results across resolutions: the proposed Monte Carlo-type Neural Operator (MCNO) outperforms FNO, MGNO, LNO, and GNO, capturing the nonlinear dynamics effectively. While MWT Leg achieves slightly lower errors, MCNO balances accuracy and efficiency, with relative L_2 loss and per-epoch computation remaining stable as the number of samples increases (Figures 4 and 5).

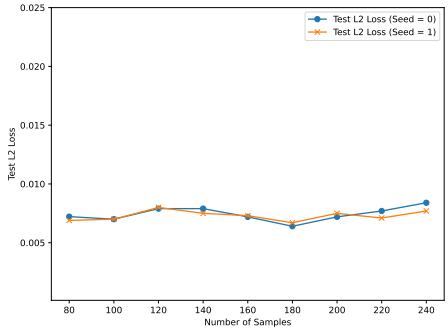

Figure 2: Relative L_2 test loss vs Number of samples for Burgers

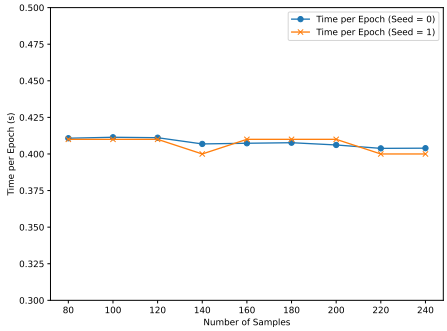

Figure 3: Time per epoch vs Number of samples for Burgers

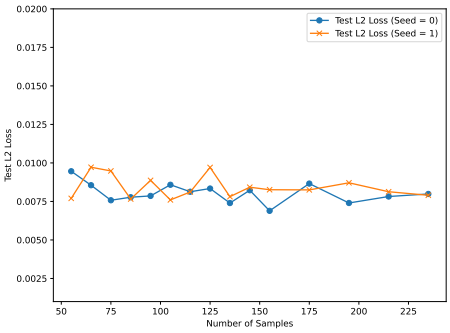

Figure 4: Relative L_2 test loss vs Number of samples for KdV

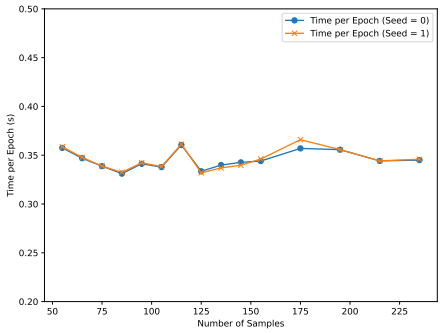

Figure 5: Time per epoch vs Number of samples for KdV

Table 2: Korteweg-de Vries (KdV) equation benchmarks showing relative L_2 errors for different input resolutions s.

Model	Time per epoch (s=128, s=1024)	$s = 128$	$s = 256$	$s = 512$	$s = 1024$
MWT Leg	(3.24s, 4.96s)	0.0036	0.0042	0.0042	0.0040
MCNO	(0.36s, 0.49s)	0.0070	0.0079	0.0088	0.0081
FNO	(0.48s, 0.50s)	0.0126	0.0125	0.0122	0.0133
MGNO	-	0.1515	0.1355	0.1345	0.1363
LNO	-	0.0557	0.0414	0.0425	0.0447
GNO	-	0.0760	0.0695	0.0699	0.0721

5 Conclusion

We introduced the Monte Carlo-type Neural Operator (MCNO), a lightweight framework that learns kernel functions using a single Monte Carlo sample to approximate integral operators. Unlike spectral or hierarchical approaches, MCNO does not assume translation invariance or rely on global bases, offering flexibility across grid resolutions and problem settings. Experiments on one-dimensional PDEs, including Burgers' and KdV equations, demonstrate that MCNO achieves competitive accuracy with computational efficiency and architectural simplicity. These results highlight MCNO as a practical alternative to established neural operator models. Future work includes extending MCNO to higher-dimensional PDEs, exploring adaptive sampling strategies, and applying it to unstructured or heterogeneous domains.

Acknowledgment

This work was supported by the NYUAD Center for Interacting Urban Networks (CITIES), funded by Tamkeen under the NYUAD Research Institute Award CG001. The views expressed in this article are those of the authors and do not reflect the opinions of CITIES or their funding agencies.

References

[1] Hitesh Gupta, Kaushik Bhattacharya, and Anima Anandkumar. Multiwavelet-based operator learning for differential equations. *Proceedings of Machine Learning Research (PMLR)*, 139:12144–12155, 2021.

[2] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to partial differential equations. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[3] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *arXiv preprint arXiv:2009.01938*, 2020.

[4] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[5] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *International Conference on Learning Representations (ICLR)*, 2021.

[6] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[7] Tapas Tripura and Souvik Chakraborty. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023.