# Solving Poisson Equations using Neural Walk-on-Spheres

**Hong Chul Nam** [*1] **Julius Berner** [*2] **Anima Anandkumar** [2]

## Abstract

We propose *Neural Walk-on-Spheres* (NWoS), a novel neural PDE solver for the efficient solution of high-dimensional Poisson equations. Leveraging stochastic representations and Walk-on-Spheres methods, we develop novel losses for neural networks based on the recursive solution of Poisson equations on spheres inside the domain. The resulting method is highly parallelizable and does not require spatial gradients for the loss. We provide a comprehensive comparison against competing methods based on PINNs, the Deep Ritz method, and (backward) stochastic differential equations. In several challenging, high-dimensional numerical examples, we demonstrate the superiority of NWoS in accuracy, speed, and computational costs. Compared to commonly used PINNs, our approach can reduce memory usage and errors by orders of magnitude. Furthermore, we apply NWoS to problems in PDE-constrained optimization and molecular dynamics to show its efficiency in practical applications.

## 1. Introduction

Partial Differential Equations (PDE) are foundational to our modern scientific understanding in a wide range of domains. While decades of research have been devoted to this topic, numerical methods to solve PDEs remain expensive for many PDEs. In recent years, deep learning has helped to accelerate the solution of PDEs (Azizadenesheli et al., 2023; Zhang et al., 2023b; Cuomo et al., 2022) as well as tackle PDEs, which had been entirely out of range for classical methods (Han et al., 2018; Scherbela et al., 2022; Nüsken & Richter, 2021b).

Among the biggest challenges for classical numerical PDE solvers are complex geometries and high dimensions. In particular, grid-based methods, such as finite-element, finite-volume, or finite-difference methods, scale exponentially in the underlying dimension. On the other hand, deep learning approaches have been shown to overcome this so-called *curse of dimensionality* (De Ryck & Mishra, 2022; Duan et al., 2021; Berner et al., 2020b). Corresponding algorithms are typically based on *Monte Carlo* (MC) approximations of variational formulations of PDEs.

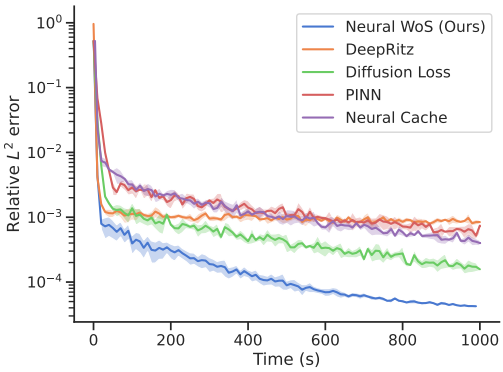


*Figure 1.* Convergence of the relative $L^2$-error when solving the $10d$ Laplace equation in Section 5 using our considered methods.

In this work, we focus on high-dimensional Poisson equations on general domains. We note that the accurate numerical solution of such types of PDEs is crucial for a large variety of areas. For instance, Poisson equations are prominent in geometry processing (Sawhney & Crane, 2020), as well as many areas of theoretical physics, e.g., electrostatics and quantum mechanics (Bahrami et al., 2014). In high dimensions, they govern important quantities in molecular dynamics, such as likely transition pathways and transition rates between regions or conformations of interest (Vanden-Eijnden et al., 2006; Lu & Nolen, 2015).

Several deep learning methods are amendable to the numerical solution of Poisson equations. This includes physics-informed neural networks (PINNs) or Deep Galerkin methods (Raissi et al., 2019; Sirignano & Spiliopoulos, 2018), the Deep Ritz method (E et al., 2017), as well as approaches based on (backward) stochastic differential equations (Han et al., 2017; Nüsken & Richter, 2021a; Han et al., 2020). However, previous methods suffer from unnecessarily high computational costs, bias, or instabilities, see Section 3.

[*]Equal contribution [1]ETH Zurich [2]Caltech. Correspondence to: Hong Chul Nam <honam@student.ethz.ch>, Julius Berner <jberner@caltech.edu>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. 

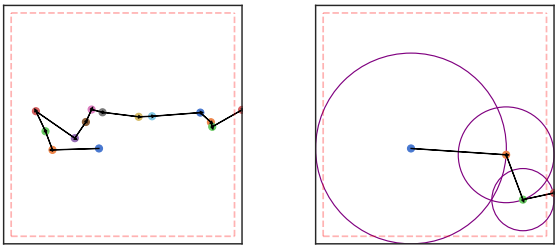

*Figure 2.* **Left:** Time-discretization of the solution $X^\xi$ to the SDE in (6) with stopping time $\tau(\Omega, \xi)$ in (7) for the domain $\Omega = [0, 1]^2$. **Right:** Realization of the Walk-on-Spheres algorithm in Section 4.

**Walk-on-Spheres (WoS):** To overcome the above challenges, we propose a novel approach based on so-called *Walk-on-Spheres* (WoS) methods (Muller, 1956). The WoS method is a Monte Carlo method specifically tailored toward Poisson equations by rewriting their solutions as an expectation over Brownian motions stopped at the boundary of the domain. Simulating the Brownian motion using time-discretizations either is slow or introduces bias (depending on the chosen time step). Leveraging the isotropy of Brownian motion, WoS accelerates this process by iteratively sampling from spheres around the current position until reaching the boundary, see Figure 2.

However, as with all Monte Carlo methods, WoS can only obtain *pointwise* estimates and suffers from slow convergence w.r.t. the number of trajectories. In particular, every sufficiently accurate estimate of the solution on a single point takes a considerable amount of time. This is prohibitive if many solution evaluations are needed sequentially, e.g., in PDE-constrained optimization problems.

**Our approach (NWoS):** We develop *Neural Walk-on-Spheres* (NWoS), a version of WoS that can be combined with neural networks to learn the solution to (parametric families of) Poisson equations on the *whole domain*. Our method amortizes the cost of WoS during training so that the solution, and its gradients, can be evaluated in fractions of seconds afterward (and at arbitrary points in the domain). In particular, in order to obtain accuracy $\varepsilon$, the standard WoS method incurs a cost of $\mathcal{O}(\varepsilon^{-2})$ trajectories for the evaluation of the solution while NWoS has a reduced cost of a single $\mathcal{O}(1)$ forward-pass of our model.

Using the partially trained model as an estimator, we can limit the number of simulations and WoS steps for training without introducing high bias or variance. The resulting objective is more efficient and scalable than competing methods, without the need to balance penalty terms for the boundary condition or compute spatial derivatives (Table 1). In particular, we demonstrate a significant reduction of GPU memory usage in comparison to PINNs (Figure 3) and up to orders of magnitude better performance for a given time and compute budget (Figure 1).

*Table 1.* Comparison of neural PDE solver for Poisson equations. *#Derivatives*, *#Loss terms*, and *Cost* denote the order of spatial derivatives, the number of terms required in the loss function, and the computational cost for one gradient step. *Propagation speed* describes how quickly boundary information can propagate to the interior of the domain, see Section 3 for details.

| Method | #Derivatives | #Loss terms | Cost | Propagation speed |
|---|---|---|---|---|
| PINN | 2 | 2 | medium | slow |
| Deep Ritz | 1 | 2 | low | slow |
| Feynman-Kac | 0 | 1 | high | fast |
| BSDE | 1 | 1 | high | fast |
| Diffusion loss | 1 | 2 | medium | medium |
| **NWoS (ours)**[1] | **0** | **1** | **low** | **fast** |

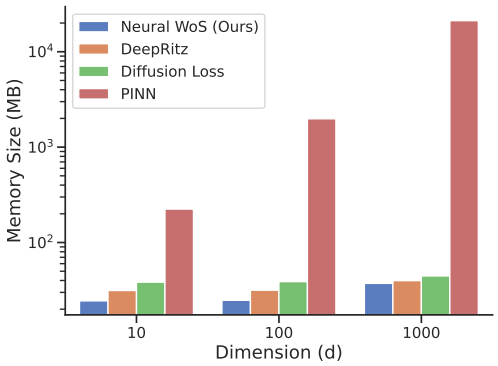


*Figure 3.* Peak GPU memory usage of different methods during training with batch size 512 for the Poisson equation in Section 5 in different dimensions $d$.

Our contributions can be summarized as follows:

- We analyze previous neural PDE solver and their shortcomings when applied to high-dimensional elliptic PDEs, such as Poisson equations (Section 3).
- We devise novel variational formulations for the solution of Poisson equations based on WoS methods and provide corresponding theoretical guarantees and efficient implementations (Section 4).
- We compare against previous approaches on a series of benchmarks and demonstrate significant improvements in terms of accuracy, speed, and scalability (Section 5).

## 2. Related works

**Neural PDE solver:** We provide an in-depth comparison to competing deep learning approaches to solve elliptic PDEs in Section 3. These include physics-informed neural networks (PINNs) (Raissi et al., 2019; Sirignano & Spiliopoulos, 2018), the Deep Ritz method (Jin et al.,

[1]While not necessary for NWoS, we note that the gradient of the model and an additional boundary loss can still be used to improve performance, see Section 4.5.

2017), and the diffusion loss (Nüsken & Richter, 2021a), see also Table 1. The diffusion loss can be viewed as an interpolation between PINNs and losses based on backward SDEs (BSDEs) (Han et al., 2017; E et al., 2017; Beck et al., 2019). Methods based on BSDEs and the Feynman-Kac formula (Beck et al., 2018; Berner et al., 2020a; Richter & Berner, 2022) have been investigated for the solution of parabolic PDEs, where the SDE is stopped at a given terminal time. Due to costly simulation times, they cannot be applied efficiently to elliptic problems. To combat this issue, we draw inspiration from Walk-on-Spheres methods.

**Monte Carlo (MC) methods:** Since grid-based methods cannot tackle high-dimensional PDEs, MC methods are typically used. For Poisson equations, the WoS method has been developed by Muller (1956) and has since been successfully used in various scientific settings (Sabelfeld, 2017; Juba et al., 2016; Bossy et al., 2010) as well as recently in computer graphics (Qi et al., 2022; Sawhney et al., 2022). In the latter domain, caches based on boundary values (Miller et al., 2023) and neural networks (Li et al., 2023) have been proposed to estimate the PDE solution across the domain and accelerate convergence. Our objective can be scaled to high-dimensional parametric PDEs and guarantees that its minimizer approximates the solution on the whole domain. We refer to Grohs & Herrmann (2022); Beznea et al. (2022) for related neural network approximation results.

## 3. Neural PDE Solver for Elliptic PDEs

We start by defining our problem and describing previous deep learning methods for its solution. Our goal is to approximate the solution[2] $u \in C(\Omega)$ to elliptic PDEs with Dirichlet boundary conditions of the form

$$\begin{cases} \mathcal{P}[u] = f, & \text{on} \quad \Omega, \\ u = g, & \text{on} \quad \partial\Omega, \end{cases} \tag{1}$$

with differential operator

$$\mathcal{P}[u] \coloneqq \tfrac{1}{2}\mathrm{Tr}(\sigma\sigma^\top \mathrm{Hess}_u) + \mu \cdot \nabla u.$$

In the above, $\Omega \subset \mathbb{R}^d$ is an open, bounded, connected, and sufficiently regular domain, see, e.g., Baldi (2017); Karatzas & Shreve (2014); Schilling & Partzsch (2014) for suitable regularity assumptions. Note that the formulation in (1) includes the Poisson equation for $\mu = 0$ and $\sigma = \sqrt{2}\mathrm{I}$, i.e.,

$$\begin{cases} \Delta u = f, & \text{on} \quad \Omega, \\ u = g, & \text{on} \quad \partial\Omega. \end{cases} \tag{2}$$

In the following, we will summarize existing neural PDE solvers for these PDEs, see Table 1 for an overview. On a high level, they propose different variational formulations $\min_{v \in V} \mathcal{L}[v]$ with the property that the minimizer over a suitable function space $V \subset C(\Omega)$ is a solution $u$ to the PDE in (1). The space $V$ is then typically approximated by a set of neural networks with a given architecture, such that the minimization problem can be tackled using variants of stochastic gradient descent.

### 3.1. Strong and weak formulations of elliptic PDEs

Let us start with methods based on strong or weak formulations of the PDE in (1).

**Physics-informed neural networks (PINNs):** In its basic form, the loss of PINNs (Raissi et al., 2019) or *Deep Galerkin* methods (Sirignano & Spiliopoulos, 2018), is given by

$$\mathcal{L}_{\mathrm{PINN}}[v] \coloneqq \mathbb{E}\left[(\mathcal{P}[v](\xi) - f(\xi))^2\right] + \beta\mathcal{L}_{\mathrm{bnd}}[v], \tag{3}$$

where

$$\mathcal{L}_{\mathrm{bnd}}[v] \coloneqq \mathbb{E}\left[(v(\zeta) - g(\zeta))^2\right]. \tag{4}$$

In the above, $\beta \in (0, \infty)$ is a penalty parameter, and $\xi$ and $\zeta$ are suitable random variables distributed on $\Omega$ and $\partial\Omega$, respectively. While improved sampling methods have been investigated, see, e.g., Tang et al. (2023); Chen et al. (2023), the default choice is to pick uniform distributions. The expectations are then approximated with standard MC or quasi-MC methods based on a suitable set of samples.

By minimizing the point-wise residual of the PDE, PINNs have gained popularity as a universal and simple method. However, PINNs are sensitive to hyperparameter choices, such as $\beta$, and suffer from training instabilities or high variance (Wang et al., 2021; Krishnapriyan et al., 2021; Nüsken & Richter, 2021b). Moreover, the objective in (3) requires the evaluation of the derivatives appearing in $\mathcal{P}[v]$. While this can be done exactly using automatic differentiation, it leads to high computational costs, see Figure 3.

**Deep Ritz method:** For the Poisson equation in (2), one can avoid this cost by leveraging weak variational formulations, see, e.g., Evans (2010). Rather than directly optimizing the regression loss in (3), the *Deep Ritz method* (E et al., 2017) proposes to minimize the objective

$$\mathcal{L}_{\mathrm{Ritz}}[v] \coloneqq \mathbb{E}\left[\frac{\|\nabla v(\xi)\|^2}{2} - f(\xi)v(\xi)\right] + \beta\mathcal{L}_{\mathrm{bnd}}[v]. \tag{5}$$

Under suitable assumptions, the minimizer again corresponds to the solution to the PDE in (2). The objective only requires computing the gradient $\nabla v$ instead of the Laplacian $\Delta v$. Using backward mode automatic differentiation, this reduces the number of backward passes from $d + 1$ to one, see also the reduced cost in Figure 3. Moreover, we note that the loss in (5) allows for weak solutions that are not

[2] For simplicity, we assume that a sufficiently smooth, strong solution exists.

twice differentiable. We refer to Chen et al. (2020), for an extensive comparison of the Deep Ritz method to PINNs for elliptic PDEs with different boundary conditions.

Finally, we mention that both methods suffer from the fact that the interior losses only consider local, pointwise information at samples $x \in \Omega$. At the beginning of training, the interior loss might thus not be meaningful. Specifically, the boundary condition $g$ first needs to be learned via the boundary loss $\mathcal{L}_{\text{bnd}}$, and then propagate from the boundary $\partial\Omega$ to interior points $x$ via the local interior loss. There exist some heuristics to mitigate this issue by, e.g., progressively learning the solution, see Penwarden et al. (2023) for an overview. The next section describes more principled ways of including boundary information in the loss and directly informing the interior points of the boundary condition.

### 3.2. Stochastic formulations of elliptic PDEs

From weak solutions, we will now proceed to stochastic representations of elliptic PDEs in (1). To this end, consider the[3] solution $X^\xi$ to the stochastic differential equation

$$\mathrm{d}X_t^\xi = \mu(X_t^\xi)\,\mathrm{d}t + \sigma(X_t^\xi)\,\mathrm{d}W_t, \quad X_0^\xi = \xi, \tag{6}$$

where $W$ is a standard $d$-dimensional Brownian motion. Moreover, we define the stopping time $\tau$ as the first exit time of the stochastic process $X^\xi$ from the domain $\Omega$, i.e.,

$$\tau = \tau(\Omega, \xi) \coloneqq \inf\{t \in [0, \infty) \colon X_t^\xi \notin \Omega\}. \tag{7}$$

An application of Itô's lemma to the process $u(X^\xi_{t\wedge\tau})$ shows that we almost surely have that

$$u(X_\tau^\xi) = u(X_0^\xi) + \int_0^\tau \mathcal{P}[u](X_t^\xi)\,\mathrm{d}t + S_\tau^u,$$

where $S_\tau^u$ is the stochastic integral

$$S_\tau^u \coloneqq \int_0^\tau (\sigma^\top \nabla u)(X_t^\xi) \cdot \mathrm{d}W_t.$$

Using the fact that $X_0^\xi = \xi$ and assuming that $u$ solves the elliptic PDE in (1), we arrive at the formula

$$g(X_\tau^\xi) = u(\xi) + F_\tau^\xi + S_\tau^u, \tag{8}$$

where we used the abbreviation

$$F_\tau^\xi \coloneqq \int_0^\tau f(X_t^\xi)\,\mathrm{d}t.$$

Since the stochastic integral $S_u^\tau$ has zero expectation, see, e.g., Baldi (2017, Theorem 10.2), we can rewrite (8) as a stochastic representation, i.e.,

$$u(x) = \mathbb{E}\left[g(X_\tau^\xi) - F_\tau^\xi \middle| \xi = x\right], \tag{9}$$

which goes back to Kakutani's Theorem (Kakutani, 1944) and is a special case of the Feynman-Kac formula.

While the representation in (9) leads to MC methods for the pointwise approximation of $u$ at a given point $x \in \Omega$, it also allows us to derive variational formulation for learning $u$ on the whole domain $\Omega$. Based on the above results, we can derive the following three losses.

**Feynman-Kac loss:** The *Feynman-Kac* loss is given by

$$\mathcal{L}_{\text{FK}}[v] \coloneqq \mathbb{E}\left[\left(v(\xi) - g(X_\tau^\xi) + F_\tau^\xi\right)^2\right] \tag{10}$$

and follows from the fact that the solution to a quadratic regression problem as in (10) is given by the conditional expectation in (9). Notably, this variational formulation does neither require a derivative of the function $v$ nor an extra boundary loss $\mathcal{L}_{\text{bnd}}$.

**BSDE loss:** Since the formula in (8) holds if and only if $u$ solves the PDE in (1), we can derive the *BSDE* loss

$$\mathcal{L}_{\text{BSDE}}[v] \coloneqq \mathbb{E}\left[\left(v(\xi) - g(X_\tau^\xi) + F_\tau^\xi + S_\tau^v\right)^2\right]. \tag{11}$$

Compared to the Feynman-Kac loss in (10), the BSDE loss requires computing the gradient of $v$ at every time-discretization of the SDE $X^\xi$ in order to compute $S_\tau^v$. However, due to (8), $S_\tau^v$ acts as a control variates and causes the variance of the MC estimator of (11) to vanish at the optimum, see Richter & Berner (2022) for details.

For the previous two losses, boundary information is directly propagated along the trajectory of the SDE $X^\xi$ to the interior. However, simulating a batch of realizations of the SDE until they reach the boundary $\partial\Omega$, i.e., until the stopping time $\tau$, can incur prohibitively high costs.

**Diffusion loss:** The *diffusion loss* (Nüsken & Richter, 2021a) circumvents long simulation times by stopping the SDE at $s = \tau \wedge T$, i.e., at the minimum of a prescribed time $T \in (0, \infty)$ and the stopping time $\tau$. Since the trajectories might not reach the boundary, the loss is supplemented with a boundary loss. This yields the variational formulation

$$\mathcal{L}_{\text{Diff}}[v] \coloneqq \mathbb{E}\left[\left(v(\xi) - v(X_s^\xi) + F_s^\xi + S_s^v\right)^2\right] + \beta\mathcal{L}_{\text{bnd}}[v].$$

Note that this can be viewed as an interpolation between the BSDE loss (for $s \to \infty$) and the PINN loss (for $s \to 0$ and rescaling by $s^{-2}$). In the same way, it also balances the advantages and disadvantages of both losses, see also Table 1.

## 4. Neural Walk-on-Spheres (NWoS) Method

In this section, we will present a more efficient way of simulating the SDE trajectories for the case of Poisson-type PDEs as in (2). Our loss is based on the FK loss in (10), which does not require the computation of any spatial derivatives

[3] We assume that there exists a unique solution, see, e.g., Le Gall (2016) for corresponding conditions.

of the neural network $v$. However, we reduce the number of steps for simulating the process $X^\xi$ while still reaching the boundary (different from the diffusion loss).

### 4.1. Recursion of elliptic PDEs on sub-domains

First, we outline how to cast the solution of the PDE in (1) into nested subproblems of solving elliptic PDEs on sub-domains. Specifically, let $\Omega_0 \subset \Omega$ be an open sub-domain containing[4] $\xi_0 := \xi$ and let $\tau_0 := \tau(\Omega_0, \xi)$ be the corresponding stopping time, defined as in (7). Analogously to (9), we obtain that

$$u(\xi) = \mathbb{E}\left[u(X^\xi_{\tau_0}) - F^\xi_{\tau_0} \middle| \xi\right]. \tag{12}$$

Note that this is a recursive definition since the solution $u$ to the PDE in (1) appears again in the expectation. To resolve the recurrence, we define the random variable $\xi_1 \sim X^\xi_{\tau_0}$ and choose another open sub-domain $\Omega_1 \subset \Omega$ containing $\xi_1$. Considering the stopping time $\tau_1 := \tau(\Omega_1, \xi_1)$, we can calculate the value of $u$ appearing in the inner expectation

$$u(X^\xi_{\tau_0}) \sim u(\xi_1) = \mathbb{E}\left[u(X^{\xi_1}_{\tau_1}) - F^{\xi_1}_{\tau_1} \middle| \xi_1\right]$$

We can now iterate this process for $k \in \mathbb{N}$ and combine the result with (12) to obtain

$$u(\xi) = \mathbb{E}\left[g(X^\xi_\tau) - \sum_{k\geq 0} F^{\xi_k}_{\tau_k} \middle| \xi\right]. \tag{13}$$

In the above, we used the strong Markov property of the SDE solution and the tower property of the conditional expectation, see also Grohs & Herrmann (2022). This nested stochastic representation can be compared to the one in (9). The next section shows how this provides a practical algorithm that terminates in finitely many steps.

### 4.2. Walk-on-Spheres

We tackle the problem of solving the Poisson equation in (2), i.e., $\mu = 0$ and $\sigma = \sqrt{2}\mathrm{I}$. Then, the SDE in (6) is just a scaled Brownian motion starting at $\xi$. Picking $\Omega_k := B_{r_k}(\xi_k)$ to be a ball of radius $r_k \in (0, \infty)$ around $\xi_k$ in the $k$-th step, the isotropy of Brownian motion ensures that

$$\xi_{k+1} \sim X^{\xi_k}_{\tau_k} \sim \mathcal{U}(\partial B_{r_k}(\xi_k)).$$

In other words, we can just sample $\xi_{k+1}$ uniformly from a sphere of radius $r_k$ around the previous value $\xi_k$. To terminate after finitely many steps, we pick the maximal radius in each step, i.e.,

$$r_k := \operatorname{dist}(\xi_k, \partial\Omega),$$

and stop at step $\kappa$ when reaching an $\varepsilon$-shell, i.e., when $r_\kappa < \varepsilon$ for a prescribed $\varepsilon \in (0, \infty)$. This allows us to "walk" from sphere to sphere until (approximately) reaching the boundary, such that we can estimate the first term in (13). Specifically, the value $u(X^\xi_\tau)$ in (13) is approximated by the boundary value $g(\bar{\xi}_\kappa)$, where

$$\bar{\xi}_\kappa := \operatorname*{arg\,min}_{x\in\partial\Omega} \|x - \xi_\kappa\|$$

is the projection to the boundary.

We note that the bias from introducing the stopping tolerance $\varepsilon$ can be estimated as $\mathcal{O}(\varepsilon)$ (Mascagni & Hwang, 2003). Moreover, for well-behaved, e.g., convex, domains $\Omega$, the average number of steps $\kappa$ behaves like $\mathcal{O}(\log(\varepsilon^{-1}))$ (Motoo, 1959; Binder & Braverman, 2012). This shows that $\varepsilon$ can be chosen sufficiently small without incurring too much additional computational cost. We note that this leads to much faster convergence than time-discretizations of the Brownian motion. In order to have a comparable bias, we would need to take steps of size $\mathcal{O}(\varepsilon)$, requiring $\Omega(\varepsilon^{-2})$ steps to converge.

### 4.3. Source term

To compute the second term in (13), we need to accumulate values of the form

$$v(z) := \mathbb{E}\left[-F^z_{\tau(B,z)}\right] \tag{14}$$

with a given ball $B = B_r(z)$. By (9), we observe that $v$ is the solution of a Poisson equation on the ball $B$ with zero Dirichlet boundary condition evaluated at $z \in \Omega$. We can thus use classical results by Boggio (1905), see also Gazzola et al. (2010), to write the solution in terms of Green's functions. Specifically, we have that

$$v(z) = -|B_r(z)|\,\mathbb{E}[f(\gamma)G_r(\gamma, z)], \tag{15}$$

where $\gamma \sim \mathcal{U}(B_r(z))$ and

$$G_r(y,z) := \begin{cases} \frac{1}{2\pi}\log\frac{r}{\|y-z\|}, & d = 2, \\ \frac{\Gamma(d/2-1)}{4\pi^{d/2}}\left(\|y-z\|^{2-d} - r^{2-d}\right), & d > 2, \end{cases}$$

see Appendix A for further details. In practice, we can now approximate the expectation in (15) using an MC estimate.

### 4.4. Learning Problem

Based on the previous derivations, we can establish a variational formulation, where the minimizer is guaranteed to approximate the Poisson equation in (2) on the whole domain $\Omega$. Specifically, we define

$$\mathcal{L}_{\mathrm{NWoS}}[v] := \mathbb{E}\left[\left(v(\xi) - \mathrm{WoS}(\xi)\right)^2\right], \tag{16}$$

[4] Since $\xi$ is a random variable, the sub-domain $\Omega_0$ is random, and the statement is to be understood for each realization.

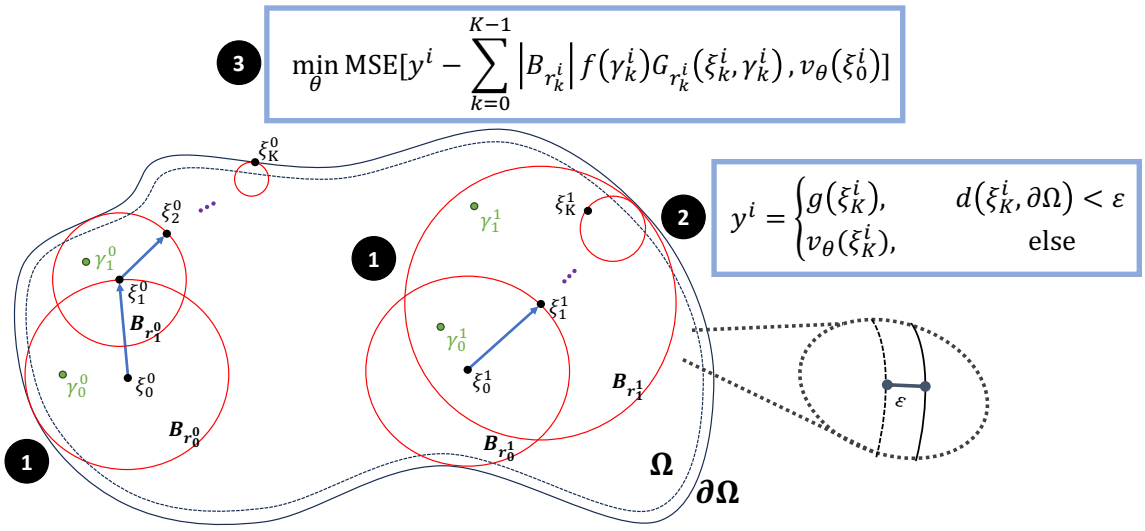


*Figure 4.* **Neural Walk-on-Spheres (NWoS):** Our algorithm for learning the solution to Poisson equations $\Delta u = f$ on $\Omega \subset \mathbb{R}^d$ and $u|_{\partial\Omega} = g$. ❶ In each gradient descent step, we sample a batch of random points $(\xi_0^i)_{i=1}^m$ in the domain $\Omega$ and simulate Brownian motions by iteratively sampling $\xi_k^i$ from spheres $B_{r_k^i}$ inscribed in the domain. To account for the source term $f$, we sample $\gamma_k^i \sim \mathcal{U}(B_{r_k^i})$ to compute an MC approximation $|B_{r_k^i}| f(\gamma_k^i) G_{r_k^i}(\xi_k^i, \gamma_k^i)$ to the solution of the Poisson equation on the sphere $B_{r_k^i}$ using the Green's function $G_{r_k^i}$ in Section 4.3. ❷ We stop after a fixed number of maximum steps $K$ and either evaluate our neural network $v_\theta$ or the boundary condition $g$ if we reach an $\varepsilon$-shell of $\partial\Omega$. ❸ If $v_\theta$ satisfies the PDE, the mean-value property implies that $v_\theta(\xi_0^i)$ is approximated by the expected value of $y^i$ minus the accumulated source term contributions. We thus minimize the corresponding mean squared error over the parameters $\theta$ using gradient descent.

where the single-trajectory WoS method $\mathrm{WoS}(\xi)$ with random initial point $\xi$ is given by

$$\mathrm{WoS}(\xi) := g(\bar{\xi}_\kappa) - \sum_{k=0}^{\kappa-1} |B_{r_k}(\xi_k)| f(\gamma_k) G_{r_k}(\gamma_k, \xi_k).$$

In the above, $\gamma_k \sim \mathcal{U}(B_{r_k}(\xi_k))$, and the random variables $\kappa$, $\xi_k$, $\bar{\xi}_\kappa$, and $r_k$ are defined as in Section 4.2. From the stochastic formulation of the solution in (13) and Proposition 3.5 in Grohs & Herrmann (2022), it follows that the minimizer of (16), i.e., $x \mapsto \mathbb{E}\left[\mathrm{WoS}(\xi) | \xi = x\right]$, approximates the solution $u$ in (2) in the uniform norm up to error $\mathcal{O}(\varepsilon)$, where $\varepsilon$ is the stopping tolerance, see Section 4.2. We also remark that, in theory, the loss requires only a single WoS trajectory per sample of $\xi$ since the minimizer of the regression problem in (16) averages out the noise.

Having established a learning problem, we can analyze both approximation and generalization errors. For the former, Grohs & Herrmann (2022) and Beznea et al. (2022) bounded the size of neural networks $v_\theta$ to approximate the solution $u$ up to a given accuracy. In particular, the number of required parameters $\theta$ only scales polynomially in the dimension $d$ and the reciprocal accuracy, as long as the functions $f$, $g$, and $\mathrm{dist}(\cdot, \partial\Omega)$ can be efficiently approximated by neural networks.

One can then leverage results by Berner et al. (2020b) to show that also the generalization error does not underlie the curse of dimensionality when minimizing the empirical risk, i.e., an MC approximation of (16), over a suitable set of neural networks $v_\theta$. Specifically, the number of required samples of $\xi$ to guarantee that the empirical minimizer approximates the solution $u$ up to a given accuracy also scales only polynomially with dimension and accuracy.

### 4.5. Implementation

In this section, we discuss implementations for the loss $\mathcal{L}_{\mathrm{NWoS}}$ in (16) described in the previous section. We summarize our algorithm in Figure 4 and provide pseudocode for the vanilla version in Algorithm 1. In the following, we present strategies to trade-off accuracy and computational cost and to reduce the variance of MC estimators. We provide pseudocode for NWoS with these improvements in Algorithm 2 and Algorithm 3 in the appendix.

**WoS with maximum number of steps:** For sufficiently regular geometries, the probability of a walk taking more than $k$ steps is exponentially decaying in $k$ (Binder & Braverman, 2012). However, if a single walk in our batch needs significantly more steps, it slows down the overall training. We thus introduce a deterministic maximum number of steps $K \in \mathbb{N}$; see Beznea et al. (2022) for a corresponding error analysis. However, we do not want to introduce non-negligible bias by, e.g., just projecting to the closest point on the boundary.

*Table 2.* Relative $L^2$-error (and standard deviations over 5 independent runs) of our considered methods, estimated using MC integration on $10^6$ uniformly distributed (unseen) points in $\Omega$.

| Method | Problem | | | |
|---|---|---|---|---|
| | Laplace ($10d$) | Committor ($10d$) | Poisson Rect. ($10d$) | Poisson ($50d$) |
| PINN | $7.42e^{-4} \pm 1.84e^{-4}$ | $4.10^{-3} \pm 1.11e^{-3}$ | $1.35e^{-2} \pm 1.57e^{-3}$ | $7.70e^{-3} \pm 2.25e^{-3}$ |
| Deep Ritz | $8.43e^{-4} \pm 6.29e^{-5}$ | $6.15e^{-3} \pm 5.30e^{-4}$ | $1.06e^{-2} \pm 6.20e^{-4}$ | $1.05e^{-3} \pm 1.70e^{-4}$ |
| Diffusion loss | $1.57e^{-4} \pm 7.74e^{-6}$ | $4.48e^{-2} \pm 6.93e^{-3}$ | $9.69e^{-2} \pm 1.03e^{-2}$ | $5.96e^{-4} \pm 1.06e^{-5}$ |
| Neural Cache | $3.99^{-4} \pm 4.08e^{-5}$ | $1.26e^{-3} \pm 5.82e^{-5}$ | $4.98e^{-2} \pm 1.80e^{-2}$ | $1.63e^{-2} \pm 1.42e^{-2}$ |
| WoS | $1.08e^{-3} \pm 1.34e^{-6}$ | $1.99e^{-3} \pm 9.79e^{-6}$ | $2.32e^{-1} \pm 2.09e^{-1}$ | $4.50e^{-3} \pm 7.38e^{-4}$ |
| **NWoS (ours)** | $\mathbf{4.29e^{-5} \pm 2.02e^{-6}}$ | $\mathbf{6.56e^{-4} \pm 2.42e^{-5}}$ | $\mathbf{2.60e^{-3} \pm 9.99e^{-5}}$ | $\mathbf{4.82e^{-4} \pm 1.32e^{-5}}$ |

Instead, we want to enforce the mean-value property on subdomains of $\Omega$ based on our recursion in Section 4.1. We thus propose to use the model $v$ instead of the boundary condition $g$ if the walk does not converge after $K$ steps, i.e., we define[5]

$$y^{\xi,v} \coloneqq \begin{cases} v(\xi_K), & d(\xi_K, \partial\Omega) > \varepsilon, \\ g(\bar{\xi}_K), & \text{else.} \end{cases}$$

We can then replace the second term in (16) by

$$\mathrm{WoS}(\xi, v) \coloneqq y^{\xi,v} - \sum_{k=0}^{K-1} |B_{r_k}(\xi_k)| f(\gamma_k) G_{r_k}(\gamma_k, \xi_k).$$

This helps to reduce the bias when $d(\xi_K, \partial\Omega)$ is non-negligible and leads to faster convergence assuming that we obtain increasingly good approximations $v_\theta \approx u$ during training of a neural network $v_\theta$. Our approach bears similarity to the diffusion loss, see Section 3; however, we do not need to use a time-discretization of the SDE.

**Boundary Loss:** We find empirically that an additional boundary loss can improve the performance of our method. While theoretically not required, it can especially help for a smaller number $K$ of maximum steps (see the previous paragraph). In general, we thus sample a fraction of the points on the boundary $\partial\Omega$ and optimize

$$\mathcal{L}_{\mathrm{NWoS}}[v] + \beta \mathcal{L}_{\mathrm{bnd}}[v],$$

where $\mathcal{L}_{\mathrm{bnd}}$ is defined[6] as in (4).

**Variance-reduction:** While not necessarily needed for the objective in (16), we can still average multiple WoS trajectories $N \in \mathbb{N}$ per sample of $\xi$ to reduce the variance. This leads to the estimator

$$\widehat{\mathcal{L}}_{\mathrm{NWoS}}[v] \coloneqq \frac{1}{m} \left( \sum_{i=1}^{m} v(\xi^i) - \frac{1}{N} \sum_{n=1}^{N} \mathrm{WoS}^n(\xi^i) \right),$$

where $\xi^i$ are i.i.d. samples of $\xi$ and $\mathrm{WoS}^n(\xi^i)$ are i.i.d. samples of $\mathrm{WoS}(\xi^i)$, i.e., $N$ trajectories with the same initial point $\xi^i$, see (16). Note that we vectorize the WoS simulations across both the initial points and the trajectories, making our NWoS method highly parallelizable and scalable to large batch sizes.

We further introduce control variates to reduce the variance of estimating $\mathrm{WoS}(x)$, where we focus on a fixed $x \in \Omega$ for ease of presentation. Control variates seek to reduce the variance by using an estimator of the form

$$\mathbb{E}\left[\mathrm{WoS}(x)\right] \approx \mathbb{E}[\delta] + \frac{1}{N} \sum_{n=1}^{N} \mathrm{WoS}^n(x) - \delta^n,$$

where $\delta^n$ are i.i.d. samples of a random variable $\delta$ with known expectation.

Motivated by Sawhney & Crane (2020), we use an approximation of the first-order term of a Taylor series of $u$ in the direction of the first WoS step. We assume that $\nabla v_\theta$ provides an increasingly accurate approximation of the gradient $\nabla u$ during training and propose to use

$$\delta^n \coloneqq \nabla v_\theta(x) \cdot (\xi_1^n - x),$$

where $\xi_1^n$ is the first step of $\mathrm{WoS}^n(x)$. In particular, $\xi_1^n \sim \mathcal{U}(\partial B_{r_1}(x))$ and thus $\mathbb{E}[\delta] = 0$ holds for any function $v_\theta$.

While we need to compute the gradient $\nabla v_\theta(x)$ for the control variate, we mention that this operation can be detached from the computational graph. In particular, we do not need to compute the derivative of $\nabla v_\theta(x)$ w.r.t. the parameters $\theta$ as is necessary for PINNs, the Deep Ritz method, the diffusion loss, and the BSDE loss. In Appendix C, we empirically show that the overhead of using the control variates is insignificant.

**Buffer:** Motivated by Li et al. (2023), we can use a buffer to cache training points

$$\left( \xi^{(i)}, \frac{1}{N} \sum_{n=1}^{N} \mathrm{WoS}^n(\xi^{(i)}) \right)_{i=1}^{B}. \tag{17}$$

Since we only update the buffer after a given number of training steps $L \in \mathbb{N}$, this accelerates the training. Note

[5] Since we stop the walk when reaching an $\varepsilon$-shell, the first condition can also be written as $K < \kappa$.

[6] Note that $\mathcal{L}_{\mathrm{bnd}}$ can be interpreted as a special case of $\mathcal{L}_{\mathrm{NWoS}}$ where the WoS method directly terminates since the initial points are sampled on the boundary.

**Algorithm 1** Training of vanilla NWoS method

**Input:** neural network $v_\theta$ with initial parameters $\theta$, optimizer method step for updating the parameters, number of iterations $T$, batch size $m$, source term $f$, boundary term $g$, stopping tolerance $\varepsilon$
**Output:** optimized parameters $\theta$

```
for k ← 0, ..., T do
    x_Ω ← sample from ξ^{⊗m}                    ▷ Sample points in Ω
    x ← x_Ω
    r ← dist(x, ∂Ω)                             ▷ Compute distances to ∂Ω
    while r > ε do
        γ ← sample from U(B_r(x))               ▷ Estimate source
        s ← s − |B_r(x)| f(γ) G_r(x, γ)
        u ← sample from U(∂B_r(x))
        x ← x + u                               ▷ Walk to next points
        r ← dist(x, ∂Ω)                         ▷ Compute distances to ∂Ω
    end while
    x ← project x to ∂Ω                         ▷ Find closest points in ∂Ω
    y_Ω ← s + g(x)                              ▷ Estimate boundary
    L̂_NWoS ← MSE(v_θ(x_Ω), y_Ω)                 ▷ NWoS loss
    θ ← step(γ, ∇_θ L̂_NWoS)                     ▷ SGD step
end for
```

that this is not possible for the other methods since they require evaluation of the current model or its gradients. In every buffer update, we average over additional trajectories, i.e., increase $N$ in (17), for a fraction of points to improve their accuracy. However, different from Li et al. (2023), we also evict a fraction of points from the buffer and replace them with WoS estimates on newly sampled points $\xi^{(i)}$ in the domain $\Omega$ to balance the diversity and accuracy of the training data in the buffer.

*Remark* 4.1. The *Neural Cache* method by Li et al. (2023) uses a related approach to accelerate WoS methods for applications in computer graphics. However, their method never replaces any point $\xi^{(i)}$ in the buffer, i.e., only updates estimates in the buffer. We observed that the model is thus prone to overfitting on the points in the buffer, especially in high dimensions, preventing it from achieving high accuracies across the domain $\Omega$.

## 5. Experiments

In this section, we compare the performance of NWoS, PINN, DeepRitz, Diffusion loss, and Neural Cache on various problems across dimensions from $10d$ to $50d$. We do not consider the FK and BSDE losses since they incur prohibitively long runtimes for simulating the SDEs with sufficient precision. To compare against the baselines, we consider benchmarks from the works proposing the Deep Ritz and diffusion losses (Jin et al., 2017; Nüsken & Richter, 2021a). For a fair comparison, we set a fixed runtime of $25d + 750$ seconds and GPU memory budget of 2GiB for training and ran a grid search over a series of hyperparameter configurations for each method. Then, we performed 5 independent runs for the best configurations w.r.t. the relative $L^2$-error. More details on the hyperparameters and our implementations[7] can be found in Appendix B.

**Laplace Equation:** The first PDE is a Laplace equation on a square domain given by

$$f(x) = 0, \quad g(x) = \textstyle\sum_{i=0}^{d/2} x_{2i}x_{2i+1}, \quad x \in \Omega = (0,1)^d.$$

To test our models, we compare against the analytic solution as $u(x) = \sum_{i=0}^{d/2} x_{2k}x_{2k+1}$. Following Jin et al. (2017), we consider the case $d = 10$.

**Poisson Equation:** Next, we consider the Poisson equation presented in Jin et al. (2017), i.e.,

$$f(x) = 2d, \quad g(x) = \textstyle\sum_{i=1}^{d} x_i^2, \quad x \in \Omega = (0,1)^d,$$

with analytic solution $u(x) = \sum_{i=1}^{d} x_i^2$. We choose $d = 50$ and present results with[8] $d \in \{100, 500\}$ in Appendix D.

**Poisson Equation with Rectangular Annulus:** We also consider a Poisson equation on a rectangular annulus $\Omega = (-1,1)^d \setminus [-c,c]^d$ with sinusoidal boundary condition and source term

$$g(x) = \frac{1}{d}\sum_{i=1}^{d} \sin(2\pi x_i), \quad f(x) = -\frac{4\pi^2}{d}\sum_{i=1}^{d} \sin(2\pi x_i).$$

We choose $c = 0.25^{\frac{1}{d}}$ and $d = 10$, and note that the analytic solution is given by $u(x) = \frac{1}{d}\sum_{i=1}^{d} \sin(2\pi x_i)$.

**Committor Function:** The fourth equation deals with *committor functions* from molecular dynamics. These functions specify likely transition pathways and transition rates between (potentially metastable) regions or conformations of interest (Vanden-Eijnden et al., 2006; Lu & Nolen, 2015). They are typically high-dimensional and known to be challenging to compute. To compare NWoS, we consider the setting in Nüsken & Richter (2021a). The task is to estimate the probability of a particle hitting the outer surface of an annulus $\Omega = \{x \in \mathbb{R}^d : a < \|x\| < b\}$ with $a, b \in (0, \infty)$, before the inner surface.

The problem can then be formulated as solving the Laplace equation given by

$$f(x) = 0, \quad g(x) = 1_{\{\|x\|=b\}}, \quad x \in \Omega.$$

For this specific $\Omega$, a reference solution can be computed as

$$u(x) = \frac{a^2 - \|x\|^{2-d}a^2}{a^2 - b^{2-d}a^2}.$$

[7] Our PyTorch code can be found at https://github.com/bizoffermark/neural_wos.

[8] While $d = 100$ is considered by Jin et al. (2017), we find that a simple projection outperforms all models in sufficiently high dimensions for this benchmark, see Appendix D.

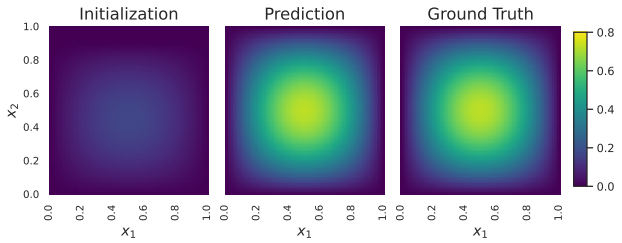


*Figure 5.* Qualitative assessment of the solution to the PDE-constrained optimization problem. **(Left)** Initial function $u_c$ for random parameters $c \in D$. **(Middle)** Predicted function $u_{\hat{c}}$ for the parameters $\hat{c}$ obtained after a few gradient descent steps using the approximation of the solution to the parametric Poisson equation obtained with NWoS. **(Right)** The groundtruth solution $u_{c^*}$.

We further use the setting by Nüsken & Richter (2021a) and choose $a = 1$, $b = 2$, and $d = 10$.

**PDE-Constrained Optimization:** Finally, we want to solve the optimization problem

$$\min_{u \in H_0^1(\Omega),\, m \in L^2(\Omega)} \quad \frac{1}{2}\int_\Omega (u - u_d)^2 dx + \frac{\alpha}{2}\int_\Omega m^2 dx$$

constraint to $u$ being a solution to the Poisson equation with $g(x) = 0$ and $f(x) = -m(x)$ for $x \in \Omega = (0,1)^2$. The goal of the optimization problem is to balance the energy of the input control $m$ with the proximity of the state $u$ and the target state $u_d$ while satisfying the PDE constraint. Following Hwang et al. (2022), we choose $u_d = \frac{1}{2\pi}\sin(\pi x_1)\sin(\pi x_2)$ as target state.

To tackle this problem and showcase the capabilities of NWoS, we first solve a parametric Poisson equation, where we parametrize the control as $m_c = c_1 \sin(c_2 x_1)\sin(c_3 x_2)$ with $c \in D := [0.5, 1.0] \times [2.5, 3.5]^2$. Similar to Berner et al. (2020a), we can sample random $c \in D$ in every gradient descent step to use NWoS for solving a whole family of Poisson equations. Freezing the trained neural network parameters afterward, we can reduce the PDE-constraint optimization problem to a problem over $c \in D$. In this illustrative example, we can compute the ground-truth parameters as $c^* = \left(\frac{1}{1+4\alpha\pi^4}, \pi, \pi\right)$ and choose $\alpha = 10^{-3}$.

### 5.1. Results

We present our results in Table 2. We first note that we improve the Deep Ritz method and the diffusion loss by almost an order of magnitude compared to the results reported by Jin et al. (2017); Nüsken & Richter (2021a). Still, our NWoS approach can outperform all other methods on our considered benchmarks. In addition to these results, we highlight that the efficient objective of NWoS also leads to faster convergence, see Figure 1. We provide ablation studies in Appendix C and additional numerical evidence in Appendix D.

The PDE-constrained optimization problem shows that NWoS can be extended to parametric problems, where a whole family of Poisson equations is solved simultaneously. We observe that for this 5-dimensional problem (two spatial dimensions and three-parameter dimensions), NWoS converges within 20 minutes to a relative $L^2$-error of $0.79\%$ (averaged over $D \times \Omega$). The trained network can then be used to solve the optimization problem directly (where we use L-BFGS) without requiring an inner loop for the PDE solver. The results show a promising relative $L^2$-error of $1.30\%$ for estimating the parameters $c^*$ leading to an accurate prediction of the minimizer, see Figure 5 and Appendix C for an ablation study.

## 6. Conclusion

We have developed Neural Walk-on-Spheres, a novel way of solving high-dimensional Poisson equations using neural networks. Specifically, we provide a variational formulation with theoretical guarantees that amortizes the cost of the standard Walk-on-Spheres algorithm to learn solutions on the full underlying domain. The resulting estimator is more efficient than competing methods (PINNs, the Deep Ritz method, and the diffusion loss) while achieving better performance at lower computational costs and faster convergence. We show that NWoS also performs better on a series of challenging, high-dimensional problems and parametric PDEs. This also highlights its potential for applications where such problems are prominent, e.g., in molecular dynamics and PDE-constraint optimization.

**Extensions and limitations:** NWoS is currently only applicable to Poisson equations with Dirichlet boundary conditions. While this PDE appears frequently in applications, we also believe that future work can extend our method. For instance, one can try to leverage adaptations of WoS to spatially varying coefficients (Sawhney et al., 2022), drift-diffusion problems (Sabelfeld, 2017), Neumann boundary conditions (Sawhney et al., 2023; Simonov, 2007), fractional Laplacians (Kyprianou et al., 2018), the screened Poisson or Helmholtz equation (Sawhney & Crane, 2020; Cheshkova, 1993), as well as linearized Poisson-Bolzmann equations (Hwang & Mascagni, 2001; Bossy et al., 2010). Moreover, one can also take other elementary shapes in each step, e.g., rectangles or stars (Deaconu & Lejay, 2006; Sawhney et al., 2023), and omit the need for $\varepsilon$-shells for certain geometries using the Green's function first-passage algorithm (Given et al., 1997).

Finally, while NWoS can tackle parametric PDEs, we need to have a fixed parametrization of the source or boundary functions. It would be promising to extend the ideas to neural operators, which currently only use losses based on PINNs (Goswami et al., 2022; Li et al., 2021) or diffusion losses for parabolic PDEs (Zhang et al., 2023a).

## Acknowledgements

The authors thank Rohan Sawhney for helpful discussions. J. Berner acknowledges support from the Wally Baer and Jeri Weiss Postdoctoral Fellowship. A. Anandkumar is supported in part by Bren endowed chair and by the AI2050 senior fellow program at Schmidt Sciences.

## Impact Statement

The aim of this work is to advance the field of machine learning and scientific computing. While there are many potential societal consequences of our work, none of them are immediate to require being specifically highlighted here.

## References

Azzizadenesheli, K., Kovachki, N., Li, Z., Liu-Schiaffini, M., Kossaifi, J., and Anandkumar, A. Neural operators for accelerating scientific simulations and design. *arXiv preprint arXiv:2309.15325*, 2023.

Bahrami, M., Großardt, A., Donadi, S., and Bassi, A. The Schrödinger–Newton equation and its foundations. *New Journal of Physics*, 16(11):115007, 2014.

Baldi, P. *Stochastic Calculus: An Introduction Through Theory and Exercises*. Universitext. Springer International Publishing, 2017.

Beck, C., Becker, S., Grohs, P., Jaafari, N., and Jentzen, A. Solving stochastic differential equations and Kolmogorov equations by means of deep learning. *arXiv preprint arXiv:1806.00421*, 2018.

Beck, C., E, W., and Jentzen, A. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, 29(4):1563–1619, 2019.

Berner, J., Dablander, M., and Grohs, P. Numerically solving parametric families of high-dimensional Kolmogorov partial differential equations via deep learning. In *Advances in Neural Information Processing Systems*, pp. 16615–16627, 2020a.

Berner, J., Grohs, P., and Jentzen, A. Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations. *SIAM Journal on Mathematics of Data Science*, 2(3):631–657, 2020b. doi: 10.1109/IWOBI.2017.7985525.

Beznea, L., Cimpean, I., Lupascu-Stamate, O., Popescu, I., and Zarnescu, A. From Monte Carlo to neural networks approximations of boundary value problems. *arXiv preprint arXiv:2209.01432*, 2022.

Binder, I. and Braverman, M. The rate of convergence of the walk on spheres algorithm. *Geometric and Functional Analysis*, 22(3):558–587, 2012.

Boggio, T. Sulle funzioni di green d'ordine m. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 20:97–135, 1905.

Bossy, M., Champagnat, N., Maire, S., and Talay, D. Probabilistic interpretation and random walk on spheres algorithms for the Poisson-Boltzmann equation in molecular dynamics. *ESAIM: Mathematical Modelling and Numerical Analysis*, 44(5):997–1048, 2010.

Chen, J., Du, R., and Wu, K. A comparison study of deep Galerkin method and deep Ritz method for elliptic problems with different boundary conditions. *arXiv preprint arXiv:2005.04554*, 2020.

Chen, X., Cen, J., and Zou, Q. Adaptive trajectories sampling for solving pdes with deep learning methods. *arXiv preprint arXiv:2303.15704*, 2023.

Cheshkova, A. "walk on spheres" algorithms for solving helmholtz equation. *Bulletin of the Novosibirsk Computing Center: Numerical analysis*, (4):7, 1993.

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

De Ryck, T. and Mishra, S. Error analysis for physics-informed neural networks (PINNs) approximating kolmogorov PDEs. *Advances in Computational Mathematics*, 48(6):1–40, 2022.

Deaconu, M. and Lejay, A. A random walk on rectangles algorithm. *Methodology and Computing in Applied Probability*, 8:135–151, 2006.

Duan, C., Jiao, Y., Lai, Y., Lu, X., and Yang, Z. Convergence rate analysis for deep ritz method. *arXiv preprint arXiv:2103.13330*, 2021.

E, W. and Yu, B. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6 (1):1–12, 2018.

E, W., Han, J., and Jentzen, A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

Evans, L. C. *Partial Differential Equations*, volume 19. American Mathematical Soc., 2010.

Gazzola, F., Grunau, H.-C., and Sweers, G. *Polyharmonic boundary value problems: positivity preserving and nonlinear higher order elliptic equations in bounded domains*. Springer Science & Business Media, 2010.

Given, J. A., Hubbard, J. B., and Douglas, J. F. A first-passage algorithm for the hydrodynamic friction and diffusion-limited reaction rate of macromolecules. *The Journal of chemical physics*, 106(9):3761–3771, 1997.

Goswami, S., Bora, A., Yu, Y., and Karniadakis, G. E. Physics-informed neural operators. *arXiv preprint arXiv:2207.05748*, 2022.

Grohs, P. and Herrmann, L. Deep neural network approximation for high-dimensional elliptic PDEs with boundary conditions. *IMA Journal of Numerical Analysis*, 42(3): 2055–2082, 2022.

Han, J., Jentzen, A., et al. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in mathematics and statistics*, 5 (4):349–380, 2017.

Han, J., Jentzen, A., and E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34): 8505–8510, 2018.

Han, J., Nica, M., and Stinchcombe, A. R. A derivative-free method for solving elliptic partial differential equations with deep neural networks. *Journal of Computational Physics*, 419:109672, 2020.

Hwang, C.-O. and Mascagni, M. Efficient modified "walk on spheres" algorithm for the linearized Poisson–Bolzmann equation. *Applied Physics Letters*, 78(6):787–789, 2001.

Hwang, R., Lee, J. Y., Shin, J. Y., and Hwang, H. J. Solving PDE-constrained control problems using operator learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4504–4512, 2022.

Jin, K. H., McCann, M. T., Froustey, E., and Unser, M. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26 (9):4509–4522, 2017.

Juba, D., Keyrouz, W., Mascagni, M., and Brady, M. Acceleration and parallelization of zeno/walk-on-spheres. *Procedia computer science*, 80:269–278, 2016.

Kakutani, S. Two-dimensional Brownian motion and harmonic functions. *Proceedings of the Imperial Academy*, 20(10):706–714, 1944.

Karatzas, I. and Shreve, S. *Brownian motion and stochastic calculus*, volume 113. springer, 2014.

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Kyprianou, A. E., Osojnik, A., and Shardlow, T. Unbiased 'walk-on-spheres' Monte Carlo methods for the fractional Laplacian. *IMA Journal of Numerical Analysis*, 38(3): 1550–1578, 2018.

Le Gall, J.-F. *Brownian motion, martingales, and stochastic calculus*. Springer, 2016.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

Li, Z., Yang, G., Deng, X., De Sa, C., Hariharan, B., and Marschner, S. Neural caches for Monte Carlo partial differential equation solvers. In *SIGGRAPH Asia 2023 Conference Papers*, pp. 1–10, 2023.

Lu, J. and Nolen, J. Reactive trajectories and the transition path process. *Probability Theory and Related Fields*, 161 (1-2):195–244, 2015.

Mascagni, M. and Hwang, C.-O. $\epsilon$-shell error analysis for "walk on spheres" algorithms. *Mathematics and computers in simulation*, 63(2):93–104, 2003.

Miller, B., Sawhney, R., Crane, K., and Gkioulekas, I. Boundary value caching for walk on spheres. *arXiv preprint arXiv:2302.11825*, 2023.

Motoo, M. Some evaluations for continuous Monte Carlo method by using brownian hitting process. *Annals of the Institute of Statistical Mathematics*, 11:49–54, 1959.

Muller, M. E. Some continuous Monte Carlo methods for the dirichlet problem. *The Annals of Mathematical Statistics*, pp. 569–589, 1956.

Nüsken, N. and Richter, L. Interpolating between BSDEs and PINNs: deep learning for elliptic and parabolic boundary value problems. *arXiv preprint arXiv:2112.03749*, 2021a.

Nüsken, N. and Richter, L. Solving high-dimensional Hamilton–Jacobi–Bellman PDEs using neural networks: perspectives from the theory of controlled diffusions and measures on path space. *Partial Differential Equations and Applications*, 2(4):1–48, 2021b.

Penwarden, M., Jagtap, A. D., Zhe, S., Karniadakis, G. E., and Kirby, R. M. A unified scalable framework for causal sweeping strategies for physics-informed neural networks (PINNs) and their temporal decompositions. *arXiv preprint arXiv:2302.14227*, 2023.

Qi, Y., Seyb, D., Bitterli, B., and Jarosz, W. A bidirectional formulation for walk on spheres. In *Computer Graphics Forum*, volume 41, pp. 51–62. Wiley Online Library, 2022.

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. *Journal of Computational Physics*, 378:686–707, 2019.

Richter, L. and Berner, J. Robust SDE-based variational formulations for solving linear PDEs via deep learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18649–18666. PMLR, 2022.

Sabelfeld, K. K. Random walk on spheres algorithm for solving transient drift-diffusion-reaction problems. *Monte Carlo Methods and Applications*, 23(3):189–212, 2017.

Sawhney, R. and Crane, K. Monte Carlo geometry processing: A grid-free approach to PDE-based methods on volumetric domains. *ACM Transactions on Graphics*, 39 (4), 2020.

Sawhney, R., Seyb, D., Jarosz, W., and Crane, K. Grid-free Monte Carlo for PDEs with spatially varying coefficients. *ACM Transactions on Graphics (TOG)*, 41(4):1–17, 2022.

Sawhney, R., Miller, B., Gkioulekas, I., and Crane, K. Walk on stars: A grid-free Monte Carlo method for PDEs with Neumann boundary conditions. *arXiv preprint arXiv:2302.11815*, 2023.

Scherbela, M., Reisenhofer, R., Gerard, L., Marquetand, P., and Grohs, P. Solving the electronic Schrödinger equation for multiple nuclear geometries with weight-sharing deep neural networks. *Nature Computational Science*, 2(5): 331–341, 2022.

Schilling, R. L. and Partzsch, L. *Brownian motion: an introduction to stochastic processes*. Walter de Gruyter GmbH & Co KG, 2014.

Simonov, N. Random walk-on-spheres algorithms for solving mixed and Neumann boundary-value problems. *Sibirskii Zhurnal Vychislitel'noi Matematiki*, 10(2):209–220, 2007.

Sirignano, J. and Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Tang, K., Wan, X., and Yang, C. DAS-PINNs: A deep adaptive sampling method for solving high-dimensional partial differential equations. *Journal of Computational Physics*, 476:111868, 2023.

Vanden-Eijnden, E. et al. Towards a theory of transition paths. *Journal of statistical physics*, 123(3):503–523, 2006.

Wang, S., Teng, Y., and Perdikaris, P. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

Zhang, R., Meng, Q., Zhu, R., Wang, Y., Shi, W., Zhang, S., Ma, Z.-M., and Liu, T.-Y. Monte Carlo neural operator for learning pdes via probabilistic representation. *arXiv preprint arXiv:2302.05104*, 2023a.

Zhang, X., Wang, L., Helwig, J., Luo, Y., Fu, C., Xie, Y., Liu, M., Lin, Y., Xu, Z., Yan, K., et al. Artificial intelligence for science in quantum, atomistic, and continuum systems. *arXiv preprint arXiv:2307.08423*, 2023b.

# A. Green's function for the Ball

For the sake of completeness, this section provides details on the derivation in Section 4.3. To compute integrals of the form (14), we look at a special case of a Poisson equation on a ball $B = B_r(z)$ with zero Dirichlet boundary condition, i.e.,

$$\begin{cases} \Delta v = f, & \text{on} \quad B, \\ v = 0, & \text{on} \quad \partial B. \end{cases}$$

Analogously to (9), we obtain that

$$v(z) = \mathbb{E}\left[ -\int_0^{\tau(B,z)} f(X_t^z)\,\mathrm{d}t \right], \tag{18}$$

where $\tau(B, z)$ is the corresponding stopping time, see (7). However, since we simplified the domain to a simple ball, we can write the solution in terms of Green's functions. Specifically, we have that

$$v(z) = -\int_B f(y) G_r(y, z)\,\mathrm{d}y \tag{19}$$

where

$$G_r(y,z) := \begin{cases} \frac{1}{2\pi} \log \frac{r}{\|y-z\|}, & d = 2, \\ \frac{\Gamma(d/2-1)}{4\pi^{d/2}} \left( \|y-z\|^{2-d} - r^{2-d} \right), & d > 2. \end{cases}$$

We note that (19) is equivalent to (15).

While this is a classical result by Boggio (1905), see also Gazzola et al. (2010), we will sketch a proof in the following. We consider the Laplace equation $\Delta \Phi_x = \delta_x$ for given $x \in \mathbb{R}^d$ in the distributional sense. It is well known that the fundamental solution $\Phi_x$ is given by

$$\Phi_x(y) = \begin{cases} \frac{1}{2\pi} \log \|y - x\|, & d = 2, \\ -\frac{\|y-x\|^{2-d}}{(d-2)\omega_d}, & d > 2, \end{cases}$$

where

$$\omega_d = |\partial B_1(0)| = \frac{2\pi^{d/2}}{\Gamma(d/2)} = \frac{4\pi^{d/2}}{(d-2)\Gamma(d/2-1)}$$

is the surface measure of the d-dimensional unit ball $B_1(0)$. Under suitable conditions, it further holds that the solution to (18) is given by

$$v(x) = \int_B f(y) \left( \Phi_x(y) - \phi_x(y) \right) \mathrm{d}y \tag{20}$$

for every $x \in B$, where the *corrector function* $\phi_x$ satisfies the Laplace equation

$$\begin{cases} \Delta \phi_x = 0, & \text{on} \quad B, \\ \phi_x = \Phi_x, & \text{on} \quad \partial B, \end{cases}$$

see Evans (2010, Chapter 2.2). Based on (9) and the fact that $\Phi_z$ is constant at the boundary of $B = z + B_r(0)$, we can compute the value of the corrector function $\phi_z$, i.e.,

$$\phi_z(y) = \mathbb{E}[\Phi_z(X^y_{\tau(B,y)})] = \begin{cases} \frac{1}{2\pi} \log r, & d = 2, \\ -\frac{r^{d-2}}{(2-d)\omega_d}, & d > 2. \end{cases}$$

This shows that the value of the Green's function at the center $z$ of the ball $B$ is given by

$$\Phi_z(y) - \phi_z(y) = -G_r(y, z),$$

which, together with (20), establishes the claim.

## A.1. Stable Implementation

For numerical stability, we directly compute the quantity $\tilde{G}_r(\gamma, z) := |B_r(z)| G_r(\gamma, z)$ in practice, as needed in (15). The volume of the hyper-sphere $|B_r(z)|$ is given by

$$|B_r(z)| = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} r^d,$$

such that we obtain

$$\tilde{G}_r(\gamma, z) :=:= \begin{cases} \frac{r^2}{2} \log \frac{r}{\|\gamma - z\|}, & d = 2, \\ \frac{r^d}{d(d-2)} \left( \|\gamma - z\|^{2-d} - r^{2-d} \right), & d > 2. \end{cases}$$

# B. Implementation Details

We implemented all methods in PyTorch and provide pseudocode in Algorithms 2 and 3. The experiments have been conducted on A100 GPUs.

For all our training, we use the Adam optimizer and limit the runtime to $25d + 750$ seconds for a fair comparison. In every step, we sample uniformly distributed samples $(\xi, \zeta)$ in the domain $\Omega$ and on the boundary $\partial\Omega$ to approximate the expectations of the loss and boundary terms. Moreover, we employ an exponentially decaying learning rate, which reduces the initial learning rate by two orders of magnitude throughout training. We choose a feedforward neural network with residual connections, 6 layers, a width of 256, and a GELU activation function. We also perform the grid search for the boundary loss penalty term, i.e.,

$$\beta \in \{0.5, 1, 5, 50, 100, 500, 1000, 5000\}.$$

We further include the batch size $m \in \{2^i\}_{i=7}^{17}$ in our grid search. For a fair comparison, we set a fixed GPU memory budget of 2GiB for training, leading to different maximal batch-sizes depending on the method; see also Figure 3. Unless otherwise specified, $10\%$ of the batch size is used for boundary points. Moreover, we set $\varepsilon = 10^{-4}$ for all methods using an $\varepsilon$-shell. Let us detail the hyperparameter choices specific to each method in the following.

**Algorithm 2** Training of our NWoS method

**Input:** neural network $v_\theta$ with initial parameters $\theta$, optimizer method step for updating the parameters, WoS method WoS in Algorithm 3, number of iterations $T$, batch sizes $m_d$ and $m_b$ for domain and boundary points, buffer $\mathcal{B}$ of size $B$, boundary function $g$, buffer update interval $L$, boundary penalty parameter $\beta$
**Output:** optimized parameters $\theta$

$x_{\partial\Omega} \leftarrow$ sample from $\zeta^{\otimes B}$ ▷ Sample points in $\partial\Omega$
$\mathcal{B} \leftarrow$ initialize with $(x_{\partial\Omega}, g(x_{\partial\Omega}))$ ▷ Initialize buffer
**for** $k \leftarrow 0, \dots, T$ **do**
  **if** $k \bmod L = 0$ **then**
    $x_\Omega \leftarrow$ sample from $\xi^{\otimes m_d}$ ▷ Sample points in $\Omega$
    $x_\mathcal{B} \leftarrow$ sample from $\mathcal{B}$ ▷ Sample points in $\mathcal{B}$
    $x \leftarrow [x_\Omega, x_\mathcal{B}]$ ▷ Concatenate points
    $[y_\Omega, y_\mathcal{B}] \leftarrow \mathrm{vmap}[\mathrm{WoS}(x, v_\theta)]$ ▷ WoS
    $\mathcal{B} \leftarrow$ update with $(x_\mathcal{B}, y_\mathcal{B})$ ▷ Update estimates
    $\mathcal{B} \leftarrow$ replace with $(x_\Omega, y_\Omega)$ ▷ Replace points
  **end if**
  $x_{\partial\Omega} \leftarrow$ sample from $\zeta^{\otimes m_b}$ ▷ Sample points in $\partial\Omega$
  $(x_\mathcal{B}, y_\mathcal{B}) \leftarrow$ sample from $\mathcal{B}$ ▷ Sample points in $\mathcal{B}$
  $\widehat{\mathcal{L}}_{\mathrm{NWoS}} \leftarrow \mathrm{MSE}(v_\theta(x_\mathcal{B}), y_\mathcal{B})$ ▷ Domain loss
  $\widehat{\mathcal{L}}_{\mathrm{bnd}} \leftarrow \mathrm{MSE}(v_\theta(x_{\partial\Omega}), g(x_{\partial\Omega}))$ ▷ Boundary loss
  $\widehat{\mathcal{L}} = \widehat{\mathcal{L}}_{\mathrm{NWoS}} + \beta\widehat{\mathcal{L}}_{\mathrm{bnd}}$
  $\theta \leftarrow \mathrm{step}\left(\gamma, \nabla_\theta \widehat{\mathcal{L}}\right)$ ▷ SGD step
**end for**

**Walk-on-Spheres (WoS):** For WoS (Muller, 1956), we directly approximate the solution at the evaluation points. We batch trajectories to saturate the memory budget and present the best result for different configurations within the given runtime. Specifically, we pick the number of trajectories $N$ in the grid $\{1, 10, 100, 1000, 10000, 100000\}$ and the maximum number of steps $K$ in $\{0, 1, 10, 100, 1000\}$.

**Neural Walk-on-Spheres (NWoS):** For NWoS, we try the different extensions in Section 4.5. Specifically, we fix the buffer size $B$ to 10 times that of the batch size $m$, and sweep the number of gradient steps between buffer updates $L \in \{10, 100, 1000\}$. We also include the maximum number of WoS steps $K \in \{0, 1, 5, 10, 50, 100\}$ and the number of trajectories per update $N \in \{1, 10, 100, 200, 300, 400, 500, 1000\}$ in our grid search. If using a boundary loss, we sweep over $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ in the grid search to find the optimal proportion of the batch size for the boundary loss.

**Neural Cache:** For the neural cache method (Li et al., 2023), we use the best configuration for different buffer sizes, update intervals, and number of trajectories within the given time and memory constraints. Specifically, we try buffer sizes $B \in \{10000, 20000, 100000, 1000000\}$, intervals $L \in \{1, 10, 100, 1000, 5000, 10000\}$ to update the buffer, and $N \in \{1, 10, 20, 30, 40, 50, 100, 500, 1000\}$ number of trajectories for each update.

**Algorithm 3** Walk-on-Spheres (WoS)

**Input:** neural network $v_\theta$, source term $f$, boundary term $g$, point for evaluation $x$, maximum number of steps $K$, stopping tolerance $\varepsilon$, number of trajectories $N$
**Output:** estimator $\widehat{v}$ of solution $v$ to PDE in (2) at $x$

$\hat{v} \leftarrow 0$
**for** $i \leftarrow 1, \dots, N$ **do** ▷ Batched in implementation
  $s \leftarrow 0$
  **for** $t \leftarrow 1, \dots, K$ **do**
    $r \leftarrow \mathrm{dist}(x, \partial\Omega)$ ▷ Compute distance to $\partial\Omega$
    **if** $r < \varepsilon$ **then**
      Break ▷ Reach boundary
    **end if**
    $\gamma \leftarrow$ sample from $\mathcal{U}(B_r(x))$ ▷ Estimate source
    $s \leftarrow s - |B_r(x)| f(\gamma) G_r(x, \gamma)$
    $u \leftarrow$ sample from $\mathcal{U}(\partial B_r(x))$
    **if** $t = 0$ & use_control_variate **then**
      $s \leftarrow s - \nabla_x v_\theta(x) \cdot u$ ▷ Control variate
    **end if**
    $x \leftarrow x + u$ ▷ Walk to next point
  **end for**
  **if** $r < \varepsilon$ **then** ▷ Estimate solution at $x$
    $x \leftarrow$ project $x$ to $\partial\Omega$ ▷ Find closest point in $\partial\Omega$
    $\widehat{v} \leftarrow s + g(x)$
  **else**
    $\widehat{v} \leftarrow s + v_\theta(x)$
  **end if**
**end for**
$\hat{v} \leftarrow \frac{1}{N}\hat{v}$ ▷ Compute MC estimate

**Diffusion loss:** For the diffusion loss (Nüsken & Richter, 2021a), we perform a grid search over the time-steps $\Delta t \in \{10^{-3}, 10^{-4}, 10^{-5}\}$ of the Euler-Maruyama scheme and the maximum number of steps in $\{1, 5, 10, 50\}$.

**PINNs:** For PINNs (Raissi et al., 2019; Sirignano & Spiliopoulos, 2018), we use automatic differentiation to compute the Laplacian $\Delta v_\theta$.

**Deep Ritz:** For the Deep Ritz method (E et al., 2017), we experiment with the original network architecture proposed in their paper. We sweep the number of blocks in $\{4, 6, 8\}$, the number of layers in $\{2, 4\}$, and the hidden dimension in $\{64, 128, 256\}$. Moreover, we replace the activation function with GELU.

## C. Ablation Studies

In this section, we provide additional ablation studies on the the contribution of our additional improvements in Section 4.5, namely the control variates as well as the neural network evaluation for trajectories that did not reach the

*Table 3.* Ablation study of the contribution of control variates and the neural network evaluation for trajectories that did not converge after a given maximum number of steps $K$. We report the relative $L^2$-error for the parameter estimation in our PDE-constrained optimization problem.

| Method | Relative $L^2$-error |
|---|---|
| Base NWoS | 2.89% |
| + Control Variate | 1.72% |
| + Terminal Eval | 1.70% |
| + Both | **1.30%** |

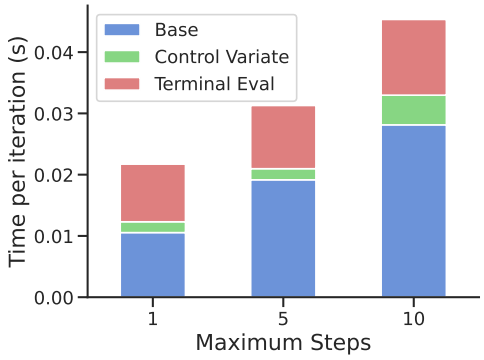


*Figure 6.* Decomposition of the training time for one iteration of NWoS in the plain version (Section 4.4), as well as using our improvements from Section 4.5, i.e., the control variates and a neural network evaluation for trajectories that did not converge after a given maximum number of steps $K$.

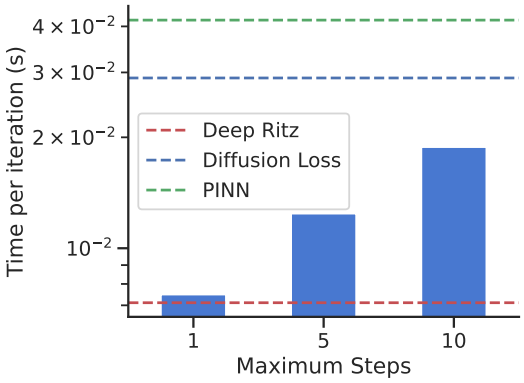


*Figure 7.* Training time of our considered methods for one gradient step. For NWoS, we present the comparison for a different maximum number of steps $K$.

boundary. We also analyze the speed of NWoS and perform comparisons with PINNs, Deep Ritz, and the diffusion loss.

In Table 3, we perform an ablation study on the contribution of our improvements in our PDE-constrained optimization problem. We observe that they can decrease the relative $L^2$-error from 2.89% to 1.72% and 1.70%, respectively. If both the control variate and the neural network evaluation are used, we obtain the best relative error of 1.30%, indicating that they can efficiently decrease bias and variance.

*Table 4.* Relative $L^2$-error (and standard deviations over 5 independent runs) of our considered methods, estimated using MC integration on $10^6$ uniformly distributed (unseen) points in $\Omega$.

| Method | Problem | |
|---|---|---|
| | Poisson ($100d$) | Poisson ($500d$) |
| PINN | $1.49e^{-3} \pm 3.21e^{-5}$ | $2.42^{-2} \pm 6.06e^{-4}$ |
| Deep Ritz | $1.77e^{-2} \pm 1.94e^{-4}$ | $9.92e^{-3} \pm 2.56e^{-5}$ |
| Diffusion loss | $6.71e{-}4 \pm 1.31e^{-5}$ | $9.47e^{-3} \pm 3.81e^{-5}$ |
| Projection | $\mathbf{2.92e^{-4} \pm 5.17e^{-7}}$ | $\mathbf{1.19e^{-5} \pm 1.67e^{-8}}$ |
| **NWoS (ours)** | $6.22e^{-4} \pm 1.18e^{-5}$ | $9.14^{-3} \pm 6.31e^{-5}$ |

Figure 6 decomposes the training time per iteration into the time for the base NWoS algorithm and the time for the additional extensions from Section 4.5. We assume the batch size to be fixed to $m = 512$ and test on the Poisson equation in Section 4.5 in $100d$. We observe that our proposed extensions incur comparably small overheads. Figure 7 further compares NWoS with DeepRitz, NSDE, and PINN with different maximum number of steps $K$, see Section 4.5. Considering the logarithmic scaling of the plot, each iteration of NWoS is significantly faster than PINN and NSDE but slower than Deep Ritz for a larger maximum number of steps $K$. Choosing $K$, we can balance high accuracy and fast training.

## D. Further Evaluations

In this section, we provide further numerical evidence. We report the convergence of the relative $L^2$-error for the other PDEs and evaluate our method on the Poisson equation in $100d$ and $500d$.

Figures 8 to 10 demonstrate that neural WoS achieves the fastest convergence in comparison to all baseline methods within the provided time and memory constraints.

Table 4 provides results for our considered methods on the Poisson equation in $100d$ and $500d$ as proposed by E & Yu (2018). We demonstrate that our NWoS method achieves lower relative $L^2$-error than the baselines. However, we discover empirically that, for this benchmark, a simple projection to the boundary achieves the highest accuracy. This can be motivated by the smoothness of the solution and the fact that uniformly distributed evaluation samples concentrate at the boundary in high dimensions.

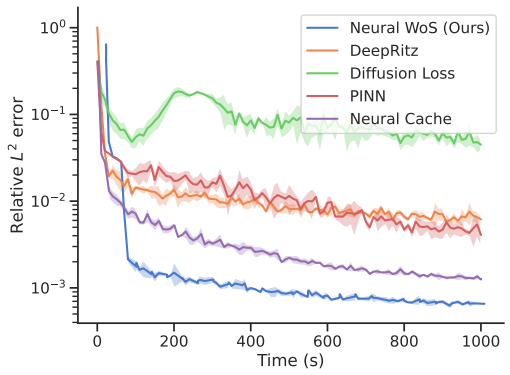


*Figure 8.* Convergence of the relative $L^2$-error when solving the Committor function in $10d$ using our considered methods.

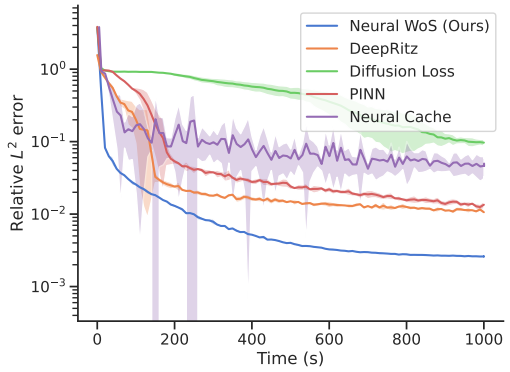


*Figure 9.* Convergence of the relative $L^2$-error when solving the Poisson equation in $10d$ with rectangular torus using our considered methods.

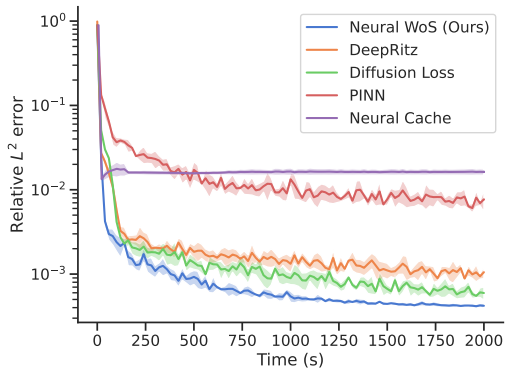


*Figure 10.* Convergence of the relative $L^2$-error when solving the Poisson equation in $50d$ using our considered methods.