Neural Control Variates with Automatic Integration

Zilu Li*
Cornell University
USA, Ithaca
zl327@cornell.edu

Guandao Yang*
Stanford University
USA, Palo Alto
guandao@stanford.edu

Qingqing Zhao
Stanford University
USA, Palo Alto
cyanzhao@stanford.edu

Xi Deng
Cornell University
USA, Ithaca
xd93@cornell.edu

Leonidas Guibas
Stanford University
USA, Palo Alto
guibas@cs.stanford.edu

Bharath Hariharan
Cornell University
USA, Ithaca
bharathh@cs.cornell.edu

Gordon Wetzstein
Stanford University
USA, Palo Alto
gordonwz@stanford.edu

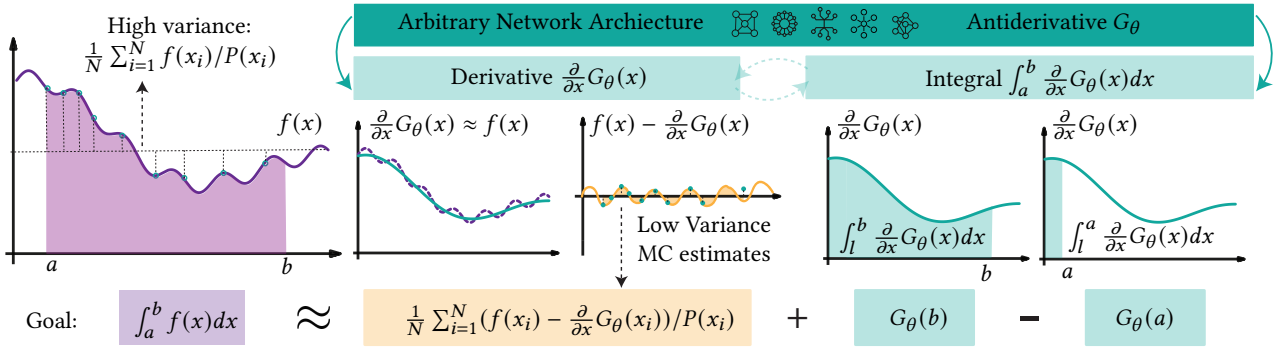

Figure 1: We propose a novel method to use arbitrary neural network architectures as control variates (CV). Instead of using the network to approximate the integrand, we deploy it to approximate the antiderivative of the integrand. This allows us to construct pairs of networks where one is the analytical integral of the other, tackling a main challenge of neural CV methods.

ABSTRACT

This paper presents a method to leverage arbitrary neural network architecture for control variates. Control variates are crucial in reducing the variance of Monte Carlo integration, but they hinge on finding a function that both correlates with the integrand and has a known analytical integral. Traditional approaches rely on heuristics to choose this function, which might not be expressive enough to correlate well with the integrand. Recent research alleviates this issue by modeling the integrands with a learnable parametric model, such as a neural network. However, the challenge remains in creating an expressive parametric model with a known analytical integral. This paper proposes a novel approach to construct learnable parametric control variates functions from arbitrary neural network architectures. Instead of using a network to approximate the integrand directly, we employ the network to approximate the anti-derivative of the integrand. This allows us to use automatic differentiation to create a function whose integration can be constructed by the antiderivative network. We apply our method to solve partial differential equations using the Walk-on-sphere algorithm [Sawhney and Crane 2020]. Our results indicate that this approach is unbiased using various network architectures and achieves lower variance than other control variate methods.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**; **Modeling and simulation**; **Neural networks**.

KEYWORDS

Control Variates, Monte Carlo Methods, PDE Solvers

ACM Reference Format:
Zilu Li, Guandao Yang, Qingqing Zhao, Xi Deng, Leonidas Guibas, Bharath Hariharan, and Gordon Wetzstein. 2024. Neural Control Variates with Automatic Integration. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27-August 1, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3641519.3657395

*Equal Contribution.

SIGGRAPH Conference Papers '24, July 27-August 1, 2024, Denver, CO, USA

ACM ISBN 979-8-4007-0525-0/24/07…$15.00
https://doi.org/10.1145/3641519.3657395

1 INTRODUCTION

Monte Carlo (MC) integration uses random samples to estimate the value of an integral. It is an essential tool in many computer graphics applications, including solving partial differential equations without discretization [Sawhney and Crane 2020] and rendering physically realistic images via ray tracing [Veach 1998]. While MC integration provides unbiased estimation for complicated integrals, it suffers from high variance. As a result, MC integration usually requires a significant amount of samples to produce an accurate estimate.

One common technique to reduce variance is control variates(CV). The key idea of control variates is to construct an alternative integral that have lower variance. For example, if we want to integrate a one-dimensional real value function $f : \mathbb{R} \to \mathbb{R}$, control variates leverage the following identity to construct different integral:

$$\int_l^u f(x)dx = G + \int_l^u f(x) - g(x)dx, \tag{1}$$

where l, u defines the integration domain in and G is the integral of real-value function $g : \mathbb{R} \to \mathbb{R}$ (*i.e.* $G = \int_\Omega g(x)dx$). If the integrand $f - g$ has less variance compared to the integrated f, then the right-hand side of this identity can result in an estimator that requires fewer samples to reach the same accuracy.

The key challenge of control variate is finding the appropriate g with known integral while minimizing the variance of $f - g$ under a certain sampling strategy. Traditional methods try to define the control variates g heuristically. For example, one way to choose g is by picking a part of f with a known integral, such as $\cos(x)$. These heuristically defined control variates may not correlate with the integrand f, limiting their performance. Recent research has proposed to parameterize the control variate using a learnable function g_θ and learn the parameter θ from samples of the integrands f [Müller et al. 2020; Salaün et al. 2022]. The hope is to find θ such that g_θ can closely match the shape of f, making $f - g_\theta$ low variance. Constructing an expressive parametric function g_θ with a known integral for all θ, however, remains challenging. As a result, existing works are limited in their choice of network architecture, such as sum of simple basic functions with known integral [Salaün et al. 2022] or special neural network architectures such as normalizing flows [Müller et al. 2020].

In this work, we propose a novel method to construct learnable control variate function g from almost arbitrary neural network architectures. Inspired by neural integration methods [Lindell et al. 2021], instead of using the network to model the control variate g directly, our method defines a network $G_\theta : \mathbb{R} \to \mathbb{R}$ to model the anti-derivative of g such that $\frac{\partial}{\partial x} G_\theta(x) = g(x)$. By the first fundamental theorem of calculus, we have:

$$G_\theta(u) - G_\theta(l) = \int_l^u \frac{\partial}{\partial x} G_\theta(x)dx. \tag{2}$$

This allows us to construct a learnable control variate using automatic differentiation frameworks in the following way:

$$\int_l^u f(x) = G_\theta(u) - G_\theta(l) + \int_l^u f(x) - \frac{\partial}{\partial x} G_\theta(x)dx. \tag{3}$$

Since $\frac{\partial}{\partial x} G_\theta(x)$ is just another neural network, we can use gradient based optimizer to find θ that minimizes the variance of the integrand $f(x) - \frac{\partial}{\partial x} G_\theta(x)$. This method allows us to use an arbitrary network architecture in place of G_θ, which enables a larger class of parametric functions to be useful for control variates. We hypothesize that this rich design space contains pairs of G_θ and $\frac{\partial}{\partial x} G_\theta(x)$ that are expressive and numerically stable enough to match f closely for various problems.

This paper takes the first steps to apply the abovementioned idea to reduce the variance of Monte Carlo integrations in computer graphics applications. To achieve this, we first extend the neural integration method from Lindell et al. [2021] from line integral to spatial integral with different domains, such as 2D disk and 3D sphere. Directly optimizing these networks to match the integrand can be numerically unstable. To alleviate this issue, we propose a numerically stable neural control variates estimator and provide corresponding training objectives to allow stable training. Finally, many graphics applications require solving recursive integration equations where different space locations have different integrand functions. We modulate the neural networks with spatially varying feature vectors to address this issue. We apply our method to create control variates for Walk-on-Sphere (WoS) algorithms [Sawhney and Crane 2020], which solve PDEs using Monte Carlo integration. Preliminary results show that our method can provide unbiased estimation from various network architectures. Our method can produce estimators with the lower variance than all baselines. To summarize, our paper has the following contributions:

- We propose a novel method to use neural networks with arbitrary architecture as a control variate function.
- We propose a numerically stable way to construct control variate estimators for different integration domains.
- We demonstrate the effectiveness of our method in solving Laplace and Poisson equations using WoS. Our method can outperform baselines in all settings.

2 RELATED WORK

We will focus on reviewing the most relevant papers in control variates and nueral integration techniques.

Control Variates. Control variates is an important variance reduction technique for Monte Carlo integration [Glynn and Szechtman 2002; Loh 1995; Pajot et al. 2014]. Prior works have applied control variates in many applications, including option pricing [Ech-Chafiq et al. 2021], variational inference [Geffner and Domke 2018; Wan et al. 2020], and Poisson image reconstruction [Rousselle et al. 2016]. To establish a control variate, we need to find a function with a known analytical integration while correlating the integrand function well. Most prior works usually construct the control variate heuristically [Clarberg and Akenine-Möller 2008; Kutz et al. 2017; Lafortune and Willems 1994]. Such an approach can be difficult to generalize to complex integrands. One way to circumvent such an issue is to make the control variates learnable and optimize the control variates function using samples from the integrand. For example, Salaün et al. [2022] proposed to use a polynomial-based estimator as control variate as the integration of the polynomial basis is easy to obtain. Recently, Müller et al. [2020] proposed to use normalizing flow as the control variate function since normalizing flows are guaranteed to integrate into one. Our method extends these works by expanding the choice of estimator family to a broader class of neural network architecture. Most existing works apply CV on physics-based rendering. We focus on applying CV to

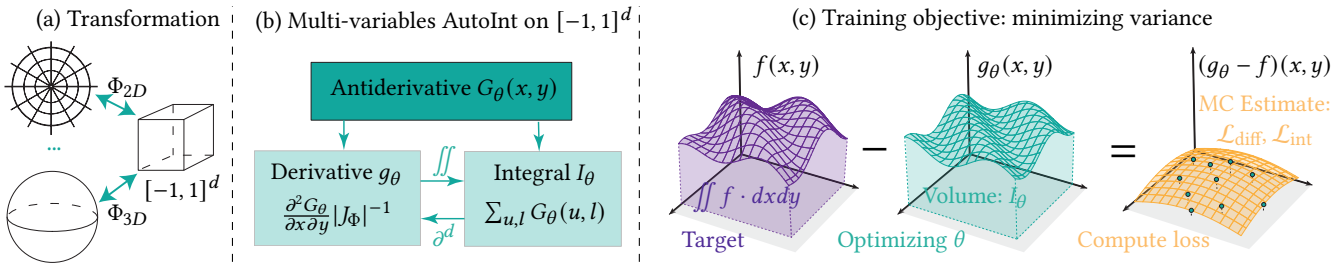

Figure 2: Overview of our method. (a) We first create a diffeomorphic transformation Φ that maps integration domain to a hyper-cube $[-1,1]^d$. (b) We generalize AutoInt [Lindell et al. 2021] to hyper-cube $[-1,1]^d$ (Sec 4.1). (c) During training, we directly minimize the variance of the estimator using Monte Carlo estimation (Sec 4.3).

solving PDEs using Walk-on-sphere methods [Sawhney and Crane 2020; Sawhney et al. 2023, 2022], which allows us to showcase the advantage of having a broader class of control variate function.

Existing works have attempted various techniques to reduce variances of the Walk-on-sphere algorithms, including caching [Bakbouk and Peers 2023; Li et al. 2023; Miller et al. 2023], heuristic-based control variates [Rioux-Lavoie et al. 2022; Sawhney and Crane 2020], and bidirectional formulations [Qi et al. 2022]. These methods are orthogonal to our paper, which applies neural control variates method to reduce variance for Walk-on-Sphere algorithms. In our experiment, we demonstrate that our method can be combined with existing variance reduction methods to reach better performance.

Neural Network Integration Methods. Deep learning has emerged as a dominant optimization tool for many applications, including numerical integration estimation. A prevalent strategy involves crafting specialized neural network architectures with analytical integration capabilities, similar in spirit to the Risch or Risch-Norman algorithm [Norman and Moore 1977; Risch 1969]. For example, normalizing flows [Chen et al. 2018; Dinh et al. 2014, 2016; Tabak and Turner 2013] is a family of network architectures that models an invertible mapping, which allows them to model probability distribution by integrating into one. Other examples include [Petrosyan et al. 2020] and [Subr 2021], which designed network architectures that can be integrated analytically. These approaches usually result in a limited choice of network architectures, which might limit the expressivity of the approximator. An alternative approach is to create computational graphs that can be integrated into a known network by taking derivatives. For example, [Nsampi et al. 2023] leverages repeated differentiation to compute convolutions of a signal represented by a network. In this work, we follow the paradigm proposed by AutoInt [Lindell et al. 2021], where we construct the integrand by taking network derivatives approximating the integration result. This approach can allow a more flexible choice of network architectures, and it has been successfully applied to other applications such as learning continuous time point processes [Zhou and Yu 2023]. Unlike the Monte Carlo integration, a potential drawback to the AutoInt method is that it can create biased estimations. In this work, we propose to combine the AutoInt method with neural control variate to create an unbiased estimator.

3 BACKGROUND

In this section, we will establish necessary notations and mathematical background to facilitate the discussion of our method. In particular, we will cover backgrounds in Monte Carlo integration, Control variates, and neural integration methods in line integration.

Monte Carlo Integration. The main idea of Monte Carlo integration is to rewrite the integration into an expectation, which can be estimated via sampling. Assume we want to estimate the integration of a real-value function $f : \mathbb{R}^d \to \mathbb{R}$ over domain Ω using Monte Carlo method. We first write it into an expectation over the domain Ω:

$$\int_{x\in\Omega} f(x)dx = \int_{x\in\Omega} P_\Omega(x)\frac{f(x)}{P_\Omega(x)}dx = \mathbb{E}_{x\sim P_\Omega}\left[\frac{f(x)}{P_\Omega(x)}\right], \quad (4)$$

where P_Ω is a distribution over domain Ω from which we can both sample and evaluate likelihood. This allows us to define the following estimator: $\langle F_N\rangle = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{P_\Omega(x_i)}$, where x_i's are points sampled from P_Ω and N denotes the number of samples. Monte Carlo estimation is unbiased given that $P_\Omega(x) = 0$ only if $f(x) = 0$. However, it usually suffers from high variance, requiring a lot of samples and function evaluations to obtain an accurate result.

Control Variates. Control variates is a technique to reduce variance for Monte Carlo estimators. The key idea is to construct a new integrand with lower variance and apply Monte Carlo estimation for the low-variance integrand only. Suppose we know $G = \int_\Omega g(x)dx$ for some G and g, then we have:

$$\int_\Omega f(x)dx = G + \int_\Omega f(x) - g(x)dx. \quad (5)$$

With this identity, we can derive a single-sample numerical estimator $\langle F_{cv}\rangle$ that is unbiased:

$$\langle F_{\text{cv}}\rangle = G + \frac{f(x_i) - g(x_i)}{P_\Omega(x_i)}, \quad \text{where } x_i \sim P_\Omega. \quad (6)$$

As long as G is the analytical integration result of g, the new estimator created after applying control variate is unbiased. Note that the control variate estimator is running Monte Carlo integration on the new integrand $f - g$, instead of the original integrand f. The key to a successful control variate is finding corresponding functions G and g that make $f - g$ to have less variance compared to the original integrand under the distribution P_Ω. Choosing an appropriate g is challenging since it requires correlation with f while having an analytical integral G. Existing works either pick g heuristically (*e.g.* $g = \cos x$ if $\cos x$ is a component of f), or use a limited family of parametric functions to approximate f from data. Our method circumvents this issue by learning a parametric

model for the antiderivative of g, allowing us to use arbitrary neural network architecture for control variate.

Neural Integration. AutoInt [Lindell et al. 2021] proposes a way to estimate an integral using neural networks. Suppose we want to estimate a line integral of the form $\int_L^U f(x)dx$, where f is a real-value function and $L \leq U \in \mathbb{R}$ denotes the lower and upper bound for integration. AutoInt trains a neural network $G_\theta : \mathbb{R} \to \mathbb{R}$ to approximate the antiderivative of the integrand f with learnable parameter θ. By the first fundamental theorem of calculus, we know that if for all x in $[L, U]$, $\frac{\partial}{\partial x} G_\theta(x) = f(x)$, then:

$$\int_L^U f(x)dx = \int_L^U \frac{\partial}{\partial x} G_\theta(x)dx = G_\theta(U) - G_\theta(L). \tag{7}$$

To find the parameter θ that satisfies the constraint $\frac{\partial}{\partial x} G_\theta(x) = f(x) \forall x \in [L, U]$, AutoInt uses gradient-based optimizer to solve the following optimization problem:

$$\theta^* = \arg\min_\theta \mathbb{E}_{x \in \mathcal{U}[L,U]} \left[\left(f(x) - \frac{\partial}{\partial x} G_\theta(x) \right)^2 \right], \tag{8}$$

where $\mathcal{U}[L, U]$ is uniformly distributed over interval $[L, U]$ and the derivative $\frac{\partial}{\partial x} G_\theta(x)$ is obtained via the automatic differentiation framework. Once the network is trained, we can use optimized parameters θ^* to approximate the integration results of $\int_L^U f(x)dx$. This idea can be extended to multi-variables integration [Maître and Santos-Mateos 2023] by taking multiple derivatives, which we will leverage in the following section to construct integrations for different spatial domains, such as disk and sphere.

Neural integration methods have several advantages. First, one can use arbitrary neural network architecture with this method. This allows users to leverage the latest and greatest network architectures, such as SIREN [Sitzmann et al. 2020] and instant-NGP [Müller et al. 2022], potentially leading to better performance. Second, neural integration can approximate a family of integrals. The abovementioned example can approximate integration of the form $\int_l^u f(x)dx$ for all pairs of $l \leq u$ such that $L \leq l \leq u \leq U$. AutoInt [Lindell et al. 2021] also show that one can modulate the network G_θ to approximate a family of different integrand. However, it's difficult to guarantee that the network $\frac{\partial}{\partial x} G_\theta$ can approximate the integrand exactly as the loss is difficult to be optimized to exactly zero. In this paper, we can circumvent such an issue as we use AutoInt inside control variates. Our method can both enjoy the advantages brought by neural integration methods while maintaining the guarantee provided by monte carlo integration methods.

4 METHOD

In this section, we will demonstrate how to use neural integration method to create control variates functions from arbitrary neural network architectures. We will first demonstrate how to construct networks with known analytical spatial integrals (Sec 4.1). We then show how to create a numerically stable unbiased estimator using these networks as control variates (Section 4.2) as well as a numerically stable training objective aiming to minimize the variance of the estimator (Sec 4.3). Finally, we will discuss how to extend this formulation to multiple domains (Sec 4.4).

Table 1: Transformation and Jacobian for variable spatial domains. Note that we assume input domain $U = [-1, 1]^d$.

Domain Ω	$\Phi : U \to \Omega$	$\lvert J_\Phi \rvert$
2D Circle	$\theta \mapsto (\cos(\pi\theta), \sin(\pi\theta))$	1
2D Disk	$(r, \theta) \mapsto \frac{r+1}{2} \cdot (\cos(\pi\theta), \sin(\pi\theta))$	$\frac{r+1}{2}$
3D Sphere	$\begin{bmatrix}\theta\\ \phi\end{bmatrix} \mapsto \begin{bmatrix}\sin(\pi(\phi+1)/2)\cos(\pi(\theta+1))\\ \sin(\pi(\phi+1)/2)\sin(\pi(\theta+1))\\ \cos(\pi(\phi+1)/2)\end{bmatrix}$	$\frac{1}{2}\sin\left(\frac{\pi(\phi+1)}{2}\right)$

4.1 Neural Spatial Integration

Computer graphics applications, such as rendering and solving PDEs, usually require integrating over spatial domains such as sphere and circles. To make neural integration methods applicable to these applications, we need to adapt them to integrate over various spatial domains by applying change of variables.

Let's assume the integration domain $\Omega \subset \mathbb{R}^d$ is parameterized by an invertible function Φ mapping from a hypercube $U = [-1, 1]^d$ to Ω, *i.e.* $\Phi(U) = \Omega$. For any neural network $G_\theta : U \to \mathbb{R}$, we can apply the first fundamental theorem of calculus to obtain the following identity [Maître and Santos-Mateos 2023]:

$$\int_U \frac{\partial^d}{\partial \mathbf{u}} G_\theta(\mathbf{u})d\mathbf{u} = \underbrace{\sum_{u_1 \in \{-1,1\}} \cdots \sum_{u_d \in \{-1,1\}} G_\theta(\mathbf{u}) \prod_{i=1}^{d} u_i}_{\text{Defined as } I_\theta}, \tag{9}$$

where $\mathbf{u} = [u_1, \ldots, u_d]$ and $\frac{\partial^d}{\partial \mathbf{u}} G_\theta(\mathbf{u})$ denotes partial derivative with respect to all dimension of vector $\mathbf{u}$: $\frac{\partial^d}{\partial \mathbf{u}_1 \ldots \partial \mathbf{u}_d} G_\theta(\mathbf{u})$. Note that we can obtain both the computation graph for the integrand $\frac{\partial^d G_\theta}{\partial \mathbf{u}}$ and the right-hand-side I_θ using existing deep learning frameworks with automatic differentiation. To extend this idea to integrating over Ω, we need to apply the change of variable:

$$I_\theta = \int_U \frac{\partial^d}{\partial \mathbf{u}} G_\theta(\mathbf{u})d\mathbf{u} = \int_\Omega \frac{\partial^d}{\partial \mathbf{u}} G_\theta(\mathbf{u}) \left| J_\Phi(\mathbf{u}) \right|^{-1} d\mathbf{x}, \tag{10}$$

where $\mathbf{x}$ are coordinate in domain Ω, $\mathbf{u} = \Phi^{-1}(\mathbf{x})$, and $J_\Phi \in \mathbb{R}^{d \times d}$ is the Jacobian matrix of function Φ. Since the integrand from the right-hand-side can be obtained through automatic differentiation, we now obtain a optimizable neural network with known integral in domain Ω. This identity is true regardless of the neural network architecture. This opens up a rich class of learnable parametric functions useful for control variates. Table 1 shows Φ and $|J_\Phi|$ in three integration domains: 2D circle, 2D disk, and 3D sphere.

4.2 Control Variates Estimator

Equation 10 now allows us to construct neural networks with analytical integral for various spatial domains such as 2D circles, 2D disks, 3D spheres, and more. These neural networks can be used for neural control variates, substituting Equation 10 into Equation 5:

$$\int_\Omega f(\mathbf{x})d\mathbf{x} = I_\theta + \int_\Omega f(\mathbf{x}) - \frac{\partial^d}{\partial \mathbf{u}} G_\theta(\mathbf{u}) \left| J_\Phi(\mathbf{u}) \right|^{-1} d\mathbf{x}, \tag{11}$$

where $\mathbf{u} = \Phi^{-1}(\mathbf{x})$. Now we can create a single-sample control variates estimator $\langle F_{\text{ncv}}(\theta)\rangle$ to approximate the spatial integration:

$$\langle F_{\text{ncv}}(\theta)\rangle = I_\theta + \frac{f(\mathbf{x}_i)}{P_\Omega(\mathbf{x}_i)} - \frac{\frac{\partial^d}{\partial \mathbf{u}} G_\theta(\mathbf{u}_i)}{|J_\Phi(\mathbf{u}_i)|\, P_\Omega(\mathbf{x}_i)}, \tag{12}$$

where $\mathbf{x}_i \sim P_\Omega$ are independent samples from a distribution on the domain Ω, $P_\Omega(\mathbf{x}_i)$ is the probability density of $\mathbf{x}_i$, and $\mathbf{u}_i = \Phi^{-1}(\mathbf{x}_i)$. While $\langle F_{\text{ncv}}(\theta)\rangle$ is unbiased, it is numerically unstable when $|J_\Phi|$ takes a very small value. For example, when integrating a 2D disk, estimator $\langle F_{\text{ncv}}(\theta)\rangle$ is unstable when $\frac{r+1}{2}$ is near 0. To tackle this issue, we first change the transformation function Φ to map U to a numerically stable domain Ω_ϵ. For the case of 2D disk, we replace Φ with $\Phi_\epsilon = \Phi \circ T_\epsilon$, where $T_\epsilon(r, \theta) = (r(2-\epsilon)/2+\epsilon, \theta)$, with a small number $\epsilon \in \mathbb{R}$. The intuition behind T_ϵ is to map the domain U to a place where $\frac{r+1}{2}$ is not near zero. We can use such a transformation Φ_ϵ to create an unbiased estimator as following:

$$\langle F_{\text{ncv}}(\theta)\rangle = I_\theta + \frac{f(\mathbf{x}_i)}{P_\Omega(\mathbf{x}_i)} - \mathbb{1}_U(\mathbf{u}_i) \frac{\frac{\partial^d}{\partial \mathbf{u}} G_\theta(\mathbf{u}_i)}{\left|J_{\Phi_\epsilon}(\mathbf{u}_i)\right| P_\Omega(\mathbf{x}_i)}, \tag{13}$$

where $\mathbf{x}_i \sim P_\Omega$, $\mathbf{u}_i = \Phi_\epsilon(\mathbf{x})$, and $\mathbb{1}_U(\mathbf{u}_i)$ is an indicator function that equals to 1 if $\mathbf{u}_i \in U$ and returns 0 otherwise.

If θ is not chosen intelligently, $\langle F_{\text{ncv}}(\theta)\rangle$ can have higher variance than directly estimating the original integrand f. We will show in the upcoming section how to minimize the variance of such an estimator using deep learning tools.

4.3 Training Objectives: Minimizing Variance

Our networks can be trained with different loss functions, and one should choose the loss function that works the best depending on the specific application. In this section, we will use the estimator's variance as an example to demonstrate how to adapt a control variate loss function to train our model. Following Müller et al. [2020], the variance of the estimator $\mathbb{V}\left[\langle F_{\text{ncv}}(\theta)\rangle\right]$ in Eq. 13 is:

$$\int_\Omega \frac{\left(f(\mathbf{x}) - \mathbb{1}_U(\mathbf{u}) \frac{\partial^d G_\theta(\mathbf{u})}{\partial \mathbf{u}} |J_\Phi(\mathbf{u})|^{-1}\right)^2}{P_\Omega(\mathbf{x})} d\mathbf{x} - \left(I_\theta - \int_\Omega f(\mathbf{x}) d\mathbf{x}\right)^2,$$

where $\mathbf{u} = \Phi^{-1}(\mathbf{x})$. Directly using this variance as a loss function is infeasible since we do not have analytical solutions for the term $\int_\Omega f(\mathbf{x})d\mathbf{x}$. Since the gradient-based optimizer only requires gradient estimate to be unbiased, one way to circumvent this issue is to create a pseudo loss whose gradient is an unbiased estimator for the gradient of $\mathbb{V}\left[\langle F_{\text{ncv}}(\theta)\rangle\right]$. First, we define the following losses:

$$\mathcal{L}_{\text{int}}(\theta, \Omega) = \mathbb{E}_{\mathbf{x}\sim\mathcal{U}(\Omega)}\left[\left(f(\mathbf{x})|\Omega| - I_\theta\right)^2\right], \tag{14}$$

$$\mathcal{L}_{\text{diff}}(\theta, \Omega) = \mathbb{E}_{\mathbf{x}\sim P_\Omega}\left[\frac{\left(f(\mathbf{x}) - \mathbb{1}_U(\mathbf{u}) \frac{\partial^d G_\theta(\mathbf{u})}{\partial \mathbf{u}} |J_\Phi(\mathbf{u})|^{-1}\right)^2}{P_\Omega(\mathbf{x})^2}\right], \tag{15}$$

where $\mathcal{U}(\Omega)$ denotes uniform sampling of domain Ω. One can verify that $\nabla_\theta \mathbb{V}\left[\langle F_{\text{ncv}}(\theta)\rangle\right] = \nabla_\theta \mathcal{L}_{\text{diff}}(\theta, \Omega) - \nabla_\theta \mathcal{L}_{\text{int}}(\theta, \Omega)$ (See supplementary). Note that $\mathcal{L}_{\text{diff}}$ can numerically unstable when $|J_\Phi(\mathbf{u})|$ is very small. Since we are using Φ_ϵ in Equation 13, we can see that $\nabla_\theta \mathcal{L}_{\text{diff}} = 0$ in the region when $|J_\Phi(\mathbf{u})|$ is very small. As a result, we discard those samples during training. Note that similar techniques can be applied to other types of control variates losses.

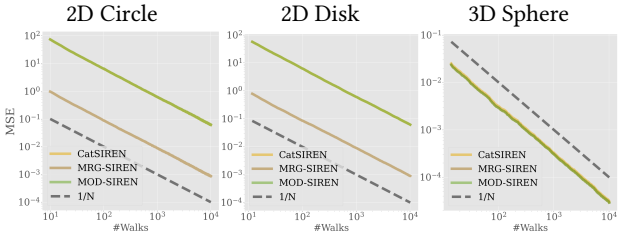

Figure 3: Convergence curve of our CV estimator using various randomly initialized networks. This suggests that our method can produce unbiased control variates estimators from arbitrary network architectures.

4.4 Modeling a Family of Integrals

So far we've focused on applying our method to a single integration $\int_\Omega f(\mathbf{x})d\mathbf{x}$ over a single domain Ω. In many computer graphics applications, we need to perform multiple spatial integrals, each of which will be using a slightly different domain and integrand. Adapting to such applications, we need to apply CV to solve a family of integrations in the form of $\int_{\Omega(\mathbf{c})} f(\mathbf{x}, \mathbf{c})d\mathbf{x}$, where $\mathbf{c} \in \mathbb{R}^h$ is a latent vector parameterizing the integral domain, $\Omega(\mathbf{c}) \subset \mathbb{R}^d$ are a set of domains changing depending on $\mathbf{c}$, and f is the integrand.

One way to circumvent this challenge is to learn coefficients for one CV that minimize the variance of all estimators as proposed by Hua et al. [2023]. In our paper, we choose an alternative way, training a conditional neural network that can predict CV functions for the whole family of integrals.

To achieve this, we first assume there exists a family of parameterization functions for this family of domains $\Phi : \mathbb{R}^d \times \mathbb{R}^h \to \Omega$, where each function $\Phi(\cdot, \mathbf{c})$ is differentiable and invertible conditional on $\mathbf{c}$, and $\Phi(U, \mathbf{c}) = \Omega(\mathbf{c})$. Now we can extend our network G_θ to take not only the integration variable $\mathbf{x}$ but also the conditioning latent vector $\mathbf{c}$. We will also extend the loss function to optimize through different latent $\mathbf{c}$: $\mathcal{L}_{\text{multi}}(\theta) = \frac{1}{N}\sum_{i=1}^N \mathcal{L}_{\text{diff}}(\theta, \Omega(\mathbf{c}_i)) - \mathcal{L}_{\text{int}}(\theta, \Omega(\mathbf{c}_i))$. The same principles described in previous sections will still apply.

5 RESULTS

In this section, we will provide a proof of concept showing that our method can be applied to reduce the variance of Walk-on-Spheres algorithms (WoS) [Sawhney and Crane 2020]. We will first demonstrate that our method creates unbiased estimators in different integration domains while using different neural network architectures (Sec 5.1). We then evaluate our method's effectiveness in solving 2D Poisson and 3D Laplace equations (Sec 5.2, Sec 5.3).

5.1 Unbiased Estimator with Arbitrary Network

In this section, we want to show that our method indeed creates an unbiased estimator regardless of neural network architectures or the integration domain. We will test our method on three types of integration domains mentioned in Table 1: 2D circle, 2D disk, and 3D spheres. We will test the following neural network architectures:

CatSIREN. Sitzmann et al. [2020] proposed SIREN, which uses periodic activation to create an expressive neural network capable of approximating high-frequency functions. We make this network architecture capable of taking conditioning, we simply concatenate

the condition vector $\mathbf{c}$ with the integration variable $\mathbf{u}$:

$$\phi_i(\mathbf{z}) = \sin(\mathbf{W}_i\mathbf{z} + \mathbf{b}_i),\ G_\theta(\mathbf{x}, \mathbf{c}) = \mathbf{W}_n(\phi_{n-1} \circ \cdots \circ \phi_0)([\mathbf{u}, \mathbf{c}]) + \mathbf{b}_n,$$

where $\theta = \{\mathbf{W}_i, \mathbf{b}_i\}_i$ are the trainable parameters.

ModSIREN. Mehta et al. [2021] proposed a way to condition SIREN network more expressively using a parallel ReLU-based network to produce the frequency of SIREN's periodic activate:

$$\mathbf{h}_0 = \max(\mathbf{W}_0'\mathbf{c} + \mathbf{b}_0', 0), \qquad \mathbf{h}_{i+1} = \max(\mathbf{W}_{i+1}'[\mathbf{c}, \mathbf{h}_i] + \mathbf{b}_{i+1}', 0)$$
$$\phi_i(\mathbf{z}) = \sin(\mathbf{W}_i\mathbf{z} \odot \mathbf{h}_i + \mathbf{b}_i), \quad G_\theta(\mathbf{x}, \mathbf{c}) = \mathbf{W}_n(\phi_{n-1} \circ \cdots \circ \phi_0)(\mathbf{x}) + \mathbf{b}_n,$$

where $\theta = \{\mathbf{W}_i, \mathbf{W}_i', \mathbf{b}_i, \mathbf{b}_i'\}_i$ are trainable parameters.

MGC-SIREN. If the conditioning latent code is low dimensional, such as $\mathbb{R}^2$ or $\mathbb{R}^3$ in our applications, we could borrow the idea from instant-NGP [Müller et al. 2022], in which we can modulate the network using a spatially varying feature grid. Basically, we use $\mathbf{c}$ to extract a latent feature $\mathbf{z}$ from a multi-resolution grid. We then use the extracted feature $\mathbf{z}$ as the conditional feature in `CatSIREN` architecture. The trainable parameters include both the feature grid and the network parameters.

In this experiment, we initialize a ground truth field $u : \mathbb{R}^d \to \mathbb{R}$ in a cube $[0, 1]^d$ and deploy these network to produce estimator for following family of integrals: $F(x) = \int_{\Omega(x)} u(x + y)dy$, where $x \in [0, 1]^2$ is a coordinate in the domain of interest and $\Omega(x)$ denotes the integration domain centered at x. For 2D circle and 2D sphere, we set $\Omega(x) = \{y \mid \|x - y\| = d(x)\}$, where $d(x)$ is the distance to the nearest point of the boundary $[0, 1]^2$. For 2D ball, we set $\Omega(x) = \{y \mid \|x - y\| \le d(x)\}$.

We randomly initialize the abovementioned three network architectures and present the MSE between the reference solution and their corresponding CV estimators (Eq 13) to ground truth. Figure 3 shows how mean square errors decay with the number of samples. Our method produces unbiased estimators for even randomly initialized neural networks with different architectures. We will stick with `MGC-SIREN` architecture for the rest of the section.

5.2 Equal Sample Comparisons

In this section, we will focus on providing equal sample analysis comparing our methods with prior arts on the task of reducing the variance of WoS algorithms in solving 2D Poisson and 3D Laplace equations. Equal sample comparisons are useful because they are less confounded with implementation and hardware details. For example, engineering techniques such as customized CUDA kernels can drastically affect the compute time on our estimator, but they won't affect the result of equal sample analysis as much.

Baselines. We compare our methods with WoS without control variates[1] and two other learning-based control variates baselines. The first baseline is Müller et al. [2020] (`NF`), which uses normalizing flows to parameterize the control variates function. The second baseline is Salaün et al. [2022] (`POLY`), which parameterizes the control variate function using a weighted sum of polynomial basis. We follow the original papers on hyperparameters, such as the degrees of the polynomial-basis and positional encoding. Specifically, we use a polynomial order of 2 following the suggestions of Fig 6 at Salaün et al. [2022]. For fair comparison, we use the same loss functions and optimizer to train the neural CV baselines. We apply multi-resolution grids mentioned in Section 5.1 to allow `POLY` baseline to perform CV at arbitrary locations within the PDE domain. Specifically, the multi-resolution grid takes as input the position of the walk (*i.e.*center of the ball/sphere), and the interpolation of the grid will provide a vector that's later decoded to the polynomial coefficients of the integrand at the grid location. We use a single linear layer to map the interpolated latent code to the set of coefficients.

[1]The original control variates method mentioned at [Sawhney and Crane 2020] can be viewed as a special case for the Salaün et al. [2022] with degree 1. As a result, we do not include it as one of the baselines.

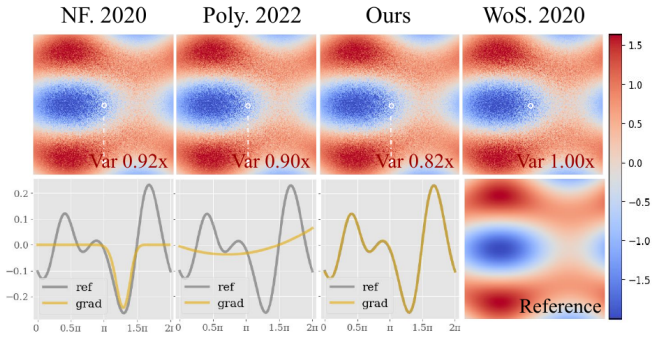

Figure 4: Top: Equal sample comparison of solving 2D Poisson equations using control variates integrating over a 2D circle. Bottom: Plotting of Network prediction and integration reference. Our network can fit the integrand tightly. This leads to an estimator that produces the lowest variance.

Training. We create a data cache of size 16384 and use samples from about 10 percent of total inference walks for training. All network is trained for 25000 optimization steps using Adam[Kingma and Ba 2014] optimizer with a learning rate of 10^{-4} and batch size 1024. For every training iteration, we update 1024 data cache locations. For each updated point, we will create a label that accumulates 32 walks to store in the data cache. All experiments are conducted with a single RTX 2080 Ti GPU.

5.2.1 Solving 2D Poisson Equation. We now apply our techniques to reduce variance on a Poisson equation over the domain $S \subset \mathbb{R}^2$:

$$\Delta u = f \text{ on } S, \quad u = g \text{ on } \partial S, \tag{16}$$

where $g : \mathbb{R}^2 \to \mathbb{R}$ is the boundary function, and $f : \mathbb{R}^2 \to \mathbb{R}$ is the forcing function. This equation can be solved by the following integral equation [Sawhney and Crane 2020]:

$$u(x) = \int_{\partial B_{d(x)}(x)} \frac{u(y)}{|\partial B_{d(x)}(x)|} dy + \int_{B_{d(x)}(x)} f(y)G(x, y)dy, \tag{17}$$

where $d(x) = \min_{y \in \partial\Omega} \|x - y\|$ denotes the distance to the boundary and $B_r(c) = \{y \| y - c| \le r\}$ is the ball centered at c with radius r. [Sawhney and Crane 2020] further derives a Monte Carlo estimator $\hat{u}(x)$ for the Poisson equation:

$$\hat{u}(x) = \begin{cases} g(x) & \text{if } d(x) < \epsilon \\ \hat{u}(x') - |B_{d(x)}(x)|f(y)G(x, y) & \text{otherwise} \end{cases}, \tag{18}$$

where $x' \sim \mathcal{U}(\partial B_{d(x_k)}(x_k)), y \sim \mathcal{U}(B_{d(x_k)}(x_k))$ are samples from the boundary and the interior of the 2D disk, and G denotes the 2D

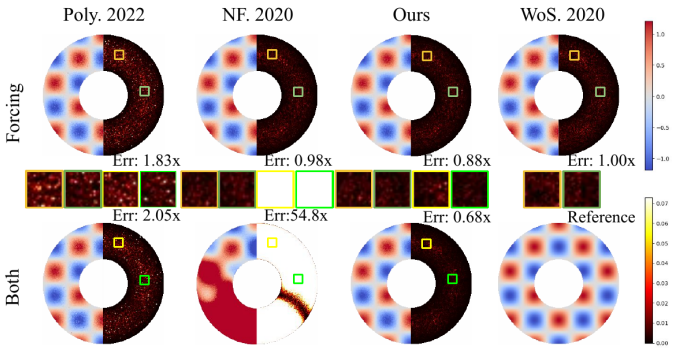

Figure 5: Equal Sample Comparison of applying control variates (CV) when solving Poisson equations within a coin domain. Top: apply CV on forcing integral; Bottom: apply CV on both forcing and recursive integral. In both settings, our method achieves the lowest error.

disk's green's function. These are two integrals that our method can be applied to: one that integrates over the circle (*i.e.* ∂B) and one that integrates inside the disk (*i.e.* B).

Apply Control Variates to 2D Circle $\partial B(x)$. We first discuss how to apply our method to reduce variance of the integral $\int_{\partial B} \frac{u(y)}{|\partial B|} dy$. We first instantiate a neural network $G_\theta(t, x)$ that takes a sample's polar angle t (normalized to $[-1, 1]$) and the 2D coordinate of the center of the disk ∂B. G_θ outputs the anti-derivative of $\frac{u(y)}{|\partial B|}$. Applying our methods, we can arrive at the following estimator:

$$\hat{u}_{\text{rec}}(x) = \begin{cases} g(x) + D(x, x') & \text{if } d(x_k) < \epsilon \\ \hat{u}_{\text{rec}}(x') + S(x, y) + D(x, x') & \text{otherwise} \end{cases}, \quad (19)$$

where t is the polar angle of vector $x' - x$, $S(x, y)$ is the single sample estimator of the forcing contribution $|B(x)|f(y)G(x, y)$, and $D(x, x') = I_\theta(x) - 2\pi \frac{\partial}{\partial t} G_\theta(t, x)$ captures the contribution of our control variates estimator. We follow the training setup described in 5.2 with $\mathcal{L}_{\text{diff}}$ loss that uses noise labels produced by $\hat{u}(x)$ to show our network can match the integrated closely. The results are presented in Figure 4.

The POLY baseline fails to produce accurate results because the high-frequency integration pattern is hard to train for polynomials with low orders. Similarly, this error also manifests in the NF baseline, even after we conducted a hyperparameter search to identify the best setting. The second row of Figure 4 shows that both NF and POLY baseline tend to produce overly smoothed control variate functions, which makes the difference between the actual and predicted integrands still a high-variance function.

On the contrary, our method can better approximate the line integral on the circle ∂B, resulting in a smaller variance.

Apply Control Variates to 2D Disk $B(x)$. Our techniques can also be applied to reducing the variance of the family of integral over 2D disks: $\int_{B_{d(x)}(x)} f(y)G(x, y)dy$. Specifically, our antiderivative network G'_θ to take a normalized polar coordinate $p \in [-1, 1]^2$ and the center of the circle $x \in \mathbb{R}^2$. Applying our method, we can construct the following single sample numerically stable control variates estimator for the forcing contribution:

$$S_{\text{cv}}(x, y) = I'_\theta(x) + |B(x)|f(x)G(x, y) - \frac{2\mathbb{1}(r)}{r+1} \frac{\partial^2}{\partial p} G'_\theta(p, y), \quad (20)$$

where $r = \|y - x\|$, $\mathbb{1}$ is an indicator guarding the numerical stability, and p is the polar coordinate of vector $y - x$. Putting this estimator inside the WoS estimator gives us an unbiased estimator:

$$\hat{u}_{\text{frc}}(x) = \begin{cases} g(x) & \text{if } d(x_k) < \epsilon \\ \hat{u}_{\text{frc}}(x') + S_{\text{cv}}(x, y) & \text{otherwise} \end{cases}. \quad (21)$$

These two control variates techniques can be combined to account for both the variance from sampling sourcing contribution as well as the variance from recursive integration:

$$\hat{u}_{\text{both}}(x) = \begin{cases} g(x) + D(x, x') & \text{if } d(x_k) < \epsilon \\ \hat{u}_{\text{both}}(x') + S_{\text{cv}}(x, y) + D(x, x') & \text{otherwise} \end{cases}. \quad (22)$$

Similarly, we use the same setup as in 5.2 and optimize the network using variance-reduction loss. We present the results of these two estimators in Figure 5. Our method consistently outperforms baselines, reaching the lowest error with the same number of samples.

5.2.2 Solving 3D Laplace Equation. In this section, we will demonstrate that our proposed method can also be applied to spherical integration. Specifically, we will apply our method to solve 3D Laplace Equation using WoS methods:

$$\Delta u = 0 \text{ on } S, \quad u = g \text{ on } \partial S, \quad (23)$$

where S is the domain where we would like to solve the Equation equation and g is the boundary condition. Similar to the Poisson equation, the Laplace equations can be solved by the estimator in Eq 18, after replacing B and ∂B with their 3D counterpart and setting the forcing function to be zero.

Applying our framework, we will train a neural network $G_\theta(s, x)$, where $s \in \mathbb{R}^2$ is a spherical coordinate normalized to $[-1, 1]^2$ and $x \in \mathbb{R}^3$ is the conditioning which modulates the integration domain $\partial B_{d(x)}(x)$. Applying our method, we can construct the following neural control variates estimator:

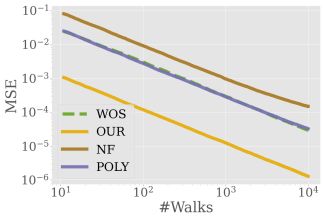

Figure 7: Our method is unbiased and achieves the lowest mean square errors.

$$\hat{u}_{\text{cv}}(x) = \begin{cases} g(x) + D(x, x') & \text{if } d(x) < \epsilon \\ \hat{u}_{\text{cv}}(x') + D(x, x') & \text{otherwise} \end{cases}, \quad (24)$$

$$D(x, x') = I_\theta(x) - \frac{\partial^2}{\partial s} G_\theta(s, x) \left(\sin\left(\frac{\phi + 1}{2} \right) \right)^{-1}, \quad (25)$$

where x' is a sample on the 3D sphere $\partial B(x)$, s is the spherical coordinate of $x' - x$, and ϕ is the polar angle of the vector $\frac{x' - x}{\|x' - x\|}$. Similar to the previous section, we obtain θ by optimizing the training objective mentioned in Section 4.4 using Adam optimizer. The training data is obtained using WoS estimator (Eq 18 with $f = 0$), which returns a noisy estimate of the integrand of interest. We present the convergence curve of our method in comparison with the baselines in Figure 7. We can see that under the same number of samples, our method achieves much lower MSE compared to

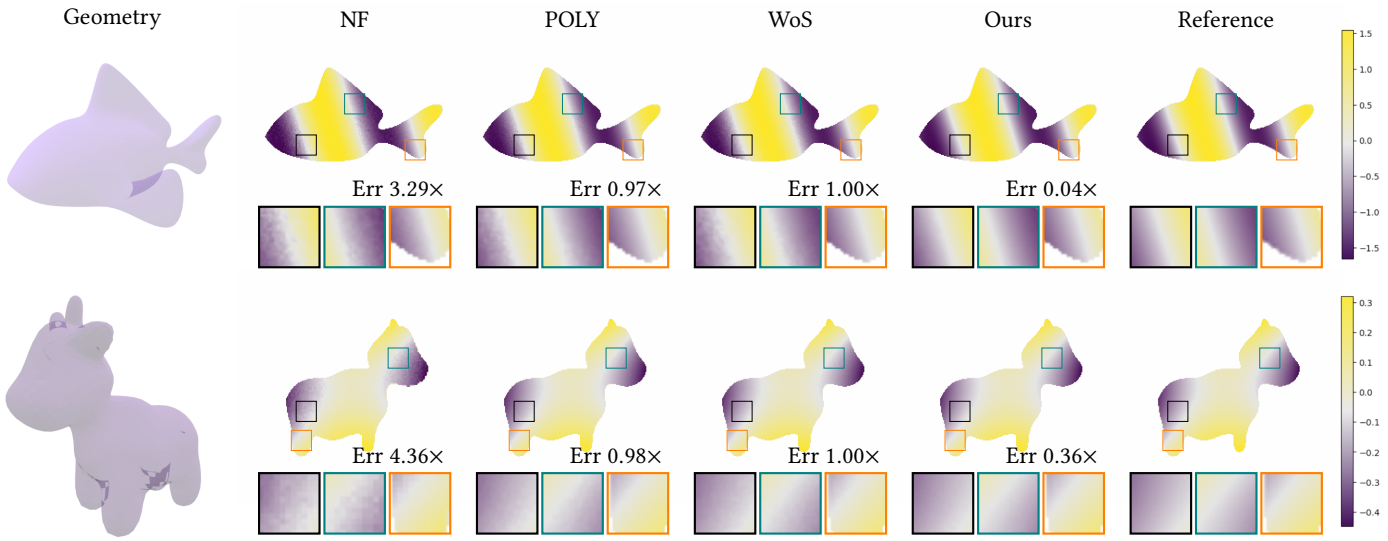

Figure 6: Equal Sample Comparison. We visualize a 2D slice of the solution to solving Laplace Equations within the Blub(top) and Spot(bottom) domains. Our method produces less noisy results and achieves lower error compared with baseline methods.

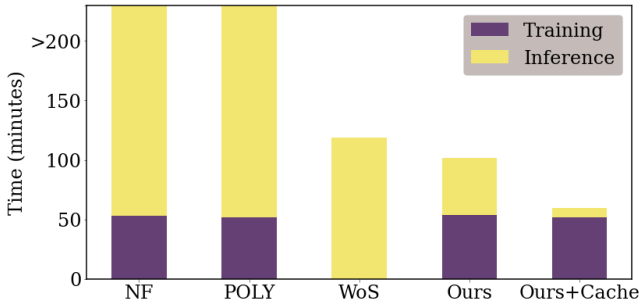

Figure 8: We report the time needed for each method to create a solution in 1024x1024 resolution with <3e-5 MSE for the Spot shape. Note that within the plotted time range, NF and POLY baselines failed to reach such MSE.

the baseline. We present the qualitative result in Figure 6 with Spot[Crane et al. 2013] and Blub[Knöppel et al. 2015]shapes and visualize a 2D slice of the solution. All results are obtained under the same number of evaluation steps. We can see that our estimator leads to a less noisy field compared to baselines.

5.3 Wall-Time Result

Despite the proposed method's advantages in equal sample comparisons, it is still a question whether such advantage can be translated to wall-time benefits since our method induces significant training and inference overheads. For example, even though generating the data required for training takes about 12 minutes, our training procedure takes much longer because it requires computing a higher-order gradient of a neural network for every iteration. Computing higher-order gradients makes each training iteration slow. Higher-order gradients also make training less stable, preventing the use of a higher learning rate; as a result, our network takes a large number of iterations to converge. Our method requires evaluating both the integral and the derivative network during inference. This means each inference step takes longer than WoS.

Table 2: MSE achieved when producing a solution of 3D Laplace equation of the Spot shape in 1024×1024 resolution within 1 hour of inference wall-time.

Methods	NF	POLY	WoS	Ours
MSE	2.4×10^{-3}	2.29×10^{-4}	5.45×10^{-5}	2.76×10^{-5}

Nonetheless, we want to demonstrate that our method can produce wall-time benefits over the baselines in applications that need to produce many very accurate queries. Specifically, we study where our method has wall-time benefits using the Blub shape in the 3D Laplace experiments (Sec 5.2.2). First, we run all methods to produce a PDE solution of 1024×1024 resolution to achieve an MSE lower than 3×10^{-5} using a RTX 2080 Ti GPU. For each method, we record the detailed compute time breakdown of training and inference time in Figure 8. Including training overhead, our method requires the least amount of time to reach the target accuracy, while other neural CV baselines fail to reach such accuracy within 200 minutes. While our method spent about half the amount of time in training, our method can reach the same accuracy with a significantly shorter inference time. This suggests that our method requires fewer walks to reach the same accuracy. As a result, our method can outperform baselines when it requires a lot of inference samples to reach the target accuracy. We also provide results on combining our method with the caching technique [Li et al. 2023] to demonstrate how such an orthogonal technique can be applied to further improve our method's wall-time efficiency (See Ours+Cache in Figure 8).

Finally, we provide an equal inference time comparison in Table 2, where we report the MSE that each method achieves given 1 hour of compute time when creating a 1024×1024 solution image. With Figure 8, it demonstrates that our method is faster than baselines in circumstances where a lot of inference samples are needed.

6 CONCLUSION AND DISCUSSION

In this paper, we propose a novel method to enable using arbitrary neural network architectures for control variates. Different from

existing methods which mostly deploy a learnable model to approximate the integrand, we ask the neural network to approximate the antiderivative of the integrand instead. The key insight is that one can use automatic differentiation to derive a network with known integral from the network that approximates the antiderivative. We apply this idea to reduce the variance of Walk-on-sphere [Sawhney and Crane 2020] Monte Carlo PDE solvers. Results suggest that our method is able to create unbiased control variates estimators from various neural network architectures and some of these networks can perform better than all baselines.

Limitations and future works. Control variates estimator usually requires more computation for each sampling step because we also need to evaluate in additional G and g for every step. This suggests that the equal-sample performance improvement might not translate to performance improvement in terms of FLOPs, wall time, or energy. In more challenging settings where the integrand f or sampling probability P is difficult to evaluate, our proposed approach might provide advantages in wall time. Computing the integration requires evaluating the antiderivative network 2^d times, where d is the integral dimension. This prevents our method from applying to higher dimensional space. One potential future direction to leverage importance sampling to improve training and inference sampling efficiency as demonstrated in Müller et al. [2020]. An other interesting direction is using these neural techniques as carriers to solve inverse problems similar to Nicolet et al. [2023].

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under grant 2144117, and by the ARL grant W911NF-21-2-0104. We want to thank Steve Marschner, Yitong Deng, Wenqi Xian, Rohan Sawhney, and George Nakayama for their feedback.

REFERENCES

Ghada Bakbouk and Pieter Peers. 2023. Mean Value Caching for Walk on Spheres. *Eurographics* (2023).

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. *Advances in neural information processing systems* 31 (2018), 1–9.

Petrik Clarberg and Tomas Akenine-Möller. 2008. Exploiting visibility correlation in direct illumination. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1125–1136.

Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2013. Robust fairing via conformal curvature flow. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Laurent Dinh, David Krueger, and Yoshua Bengio. 2014. Nice: Non-linear independent components estimation, In International Conference on Learning Representations. *International Conference on Learning Representations*.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2016. Density estimation using Real NVP. In *International Conference on Learning Representations*.

Zineb El Filali Ech-Chafiq, Jérôme Lelong, and Adil Reghai. 2021. Automatic control variates for option pricing using neural networks. *Monte Carlo Methods and Applications* 27 (2021), 91 – 104. https://api.semanticscholar.org/CorpusID:234204906

Tomas Geffner and Justin Domke. 2018. Using large ensembles of control variates for variational inference. *Advances in Neural Information Processing Systems* 31 (2018).

Peter W Glynn and Roberto Szechtman. 2002. Some new perspectives on the method of control variates. In *Monte Carlo and Quasi-Monte Carlo Methods 2000: Proceedings of a Conference held at Hong Kong Baptist University, Hong Kong SAR, China, November 27–December 1, 2000*. Springer, 27–49.

Qingqin Hua, Pascal Grittmann, and Philipp Slusallek. 2023. Revisiting controlled mixture sampling for rendering applications. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–13.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* abs/1412.6980 (2014). https://api.semanticscholar.org/CorpusID:6628106

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe Patterns on Surfaces. *ACM Trans. Graph.* 34 (2015). Issue 4.

Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. 2017. Spectral and decomposition tracking for rendering heterogeneous volumes. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–16.

Eric P Lafortune and Yves D Willems. 1994. *Using the modified phong reflectance model for physically based rendering*. Katholieke Universiteit Leuven. Departement Computerwetenschappen.

Zilu Li, Guandao Yang, Xi Deng, Christopher De Sa, Bharath Hariharan, and Steve Marschner. 2023. Neural Caches for Monte Carlo Partial Differential Equation Solvers. In *SIGGRAPH Asia 2023 Conference Papers (SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 34, 10 pages. https://doi.org/10.1145/3610548.3618141

David B Lindell, Julien NP Martel, and Gordon Wetzstein. 2021. Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14556–14565.

Wing Wah Loh. 1995. *On the method of control variates*. Stanford University.

Daniel Maître and Roi Santos-Mateos. 2023. Multi-variable integration with a neural network. *Journal of High Energy Physics* 2023, 3 (2023), 1–16.

Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. 2021. Modulated periodic activations for generalizable local functional representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 14214–14223.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2023. Boundary Value Caching for Walk on Spheres. *ACM Trans. Graph.* 42, 4, Article 82 (jul 2023), 11 pages. https://doi.org/10.1145/3592400

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. 2020. Neural control variates. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–19.

Baptiste Nicolet, Fabrice Rousselle, Jan Novak, Alexander Keller, Wenzel Jakob, and Thomas Müller. 2023. Recursive control variates for inverse rendering. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–13.

Arthur C Norman and PMA Moore. 1977. Implementing the new Risch integration algorithm. In *Proceedings of the 4th International Colloquium on Advanced Computing Methods in Theoretical Physics*. 99–110.

Ntumba Elie Nsampi, Adarsh Djeacoumar, Hans-Peter Seidel, Tobias Ritschel, and Thomas Leimkühler. 2023. Neural Field Convolutions by Repeated Differentiation. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–11.

Anthony Pajot, Loïc Barthe, and Mathias Paulin. 2014. Globally Adaptive Control Variate for Robust Numerical Integration. *SIAM Journal on Scientific Computing* 36, 4 (2014), A1708–A1730.

Armenak Petrosyan, Anton Dereventsov, and Clayton G Webster. 2020. Neural network integral representations with the ReLU activation function. In *Mathematical and Scientific Machine Learning*. PMLR, 128–143.

Yang Qi, Dario Seyb, Benedikt Bitterli, and Wojciech Jarosz. 2022. A bidirectional formulation for Walk on Spheres. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 51–62.

Damien Rioux-Lavoie, Ryusuke Sugimoto, Tümay Özdemir, Naoharu H Shimada, Christopher Batty, Derek Nowrouzezahrai, and Toshiya Hachisuka. 2022. A Monte Carlo Method for Fluid Simulation. *ACM Trans. Graph.* 41, 6 (Nov. 2022), 1–16.

Robert H. Risch. 1969. The problem of integration in finite terms. *Trans. Amer. Math. Soc.* 139 (1969), 167–189. https://api.semanticscholar.org/CorpusID:122648356

Fabrice Rousselle, Wojciech Jarosz, and Jan Novák. 2016. Image-space control variates for rendering. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–12.

Corentin Salaün, Adrien Gruson, Binh-Son Hua, Toshiya Hachisuka, and Gurprit Singh. 2022. Regression-based Monte Carlo integration. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–14.

Rohan Sawhney and Keenan Crane. 2020. Monte Carlo geometry processing. *ACM Trans. Graph.* 39, 4 (Aug. 2020).

Rohan Sawhney, Bailey Miller, Ioannis Gkioulekas, and Keenan Crane. 2023. Walk on Stars: A Grid-Free Monte Carlo Method for PDEs with Neumann Boundary Conditions. *ACM Trans. Graph.* 42, 4, Article 80 (jul 2023), 20 pages. https://doi.org/10.1145/3592398

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. 2022. Grid-free Monte Carlo for PDEs with spatially varying coefficients. *ACM Trans. Graph.* 41, 4, Article 53 (jul 2022), 17 pages. https://doi.org/10.1145/3528223.3530134

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems* 33 (2020), 7462–7473.

Kartic Subr. 2021. Q-NET: A Network for Low-dimensional Integrals of Neural Proxies. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 61–71.

Esteban G Tabak and Cristina V Turner. 2013. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics* 66, 2 (2013), 145–164.

Eric Veach. 1998. *Robust Monte Carlo methods for light transport simulation.* Stanford University.

Ruosi Wan, Mingjun Zhong, Haoyi Xiong, and Zhanxing Zhu. 2020. Neural control variates for Monte Carlo variance reduction. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II.* Springer, 533–547.

Zihao Zhou and Rose Yu. 2023. Automatic Integration for Fast and Interpretable Neural Point Processes. In *Learning for Dynamics and Control Conference.* PMLR, University of Pennsylvania, 573–585.