Learning Physics-Informed Neural Networks without Stacked Back-propagation

Di He[1]　　　　**Shanda Li**[2]　　　　**Wenlei Shi**[3]　　　　**Xiaotian Gao**[3]

Jia Zhang[3]　　　　**Jiang Bian**[3]　　　　**Liwei Wang**[1,4]　　　　**Tie-Yan Liu**[3]

[1] National Key Laboratory of General Artificial Intelligence,
School of Intelligence Science and Technology, Peking University
[2] Machine Learning Department, School of Computer Science, Carnegie Mellon University
[3] Microsoft Research　　　　[4] Center for Data Science, Peking University

Abstract

Physics-Informed Neural Network (PINN) has become a commonly used machine learning approach to solve partial differential equations (PDE). But, facing high-dimensional second-order PDE problems, PINN will suffer from severe scalability issues since its loss includes second-order derivatives, the computational cost of which will grow along with the dimension during stacked back-propagation. In this work, we develop a novel approach that can significantly accelerate the training of Physics-Informed Neural Networks. In particular, we parameterize the PDE solution by the Gaussian smoothed model and show that, derived from Stein's Identity, the second-order derivatives can be efficiently calculated without back-propagation. We further discuss the model capacity and provide variance reduction methods to address key limitations in the derivative estimation. Experimental results show that our proposed method can achieve competitive error compared to standard PINN training but is significantly faster. Our code is released at `https://github.com/LithiumDA/PINN-without-Stacked-BP`.

1 INTRODUCTION

Partial Differential Equations (PDEs) play a prominent role in describing the governing physical laws underlying a given system. Finding the solution of PDEs is important in understanding and predicting the physical phenomena from the laws. Recently, researchers sought to solve PDEs

via machine learning methods by leveraging the power of deep neural networks [Khoo et al., 2017, Han et al., 2018, Long et al., 2018, Long et al., 2019, Raissi et al., 2019, Sirignano and Spiliopoulos, 2018]. One of the seminal works in this direction is the Physics-Informed Neural Networks (PINN) approach [Raissi et al., 2019]. PINN parameterizes the solution as a neural network and learns the weights by minimizing some loss functional related to the PDEs, e.g., the PDE residual.

Although the framework is general to learn any PDEs, few previous works experimented with PINN on high-dimensional second-order PDE problems. By thorough investigation, we find PINN training suffers from a significant scalability issue, mainly resulting from stacked back-propagation. Note that the PDE residual loss contains second-order derivatives. To update the weights of the neural network by gradient descent, one must first perform automatic differentiation (i.e., back-propagation) multiple times to compute the derivatives in the PDE and then calculate the loss. For high-dimensional second-order PDEs, the computational cost in such stacked back-propagation grows along with increasing input dimension [Pang et al., 2020, Meng et al., 2021]. This will result in considerable inefficiency, making the PINN approach impractical in large-scale settings. Since some fundamental PDEs, such as Hamilton-Jacobi-Bellman Equations, are high-dimensional second-order PDEs, addressing the scalability issue of PINN becomes essential.

In this paper, we take a first step to tackle the scalability issue of PINN by developing a novel approach to train the model without stacked back-propagation. Particularly, we parameterize the PDE solution $u(x; \theta)$ as a Gaussian smoothed model, $u(x; \theta) = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 I)} f(x + \delta; \theta)$, where u transforms arbitrary base network f by injecting Gaussian noise into input x. This transformation gives rise to a key property for u where its derivatives to the input can be efficiently calculated *without back-propagation*. Such property is derived from the well-known Stein's Identity [Stein, 1981] that essentially tells that the derivatives of any

Gaussian smoothed function u can be reformulated as some expectation terms of the output of its base f, which can be estimated using Monte Carlo methods.

To be concrete, given any PDE problem, we can replace the derivative terms in the PDE with Stein's derivative estimators, calculate the (estimated) residual losses in the forward pass, and then update the weight of the parameters via one-time back-propagation. Our method can accelerate the training of PINN from two folds of advantages. First, after using Stein's derivative estimators, we no longer need stacked back-propagation to compute the loss, therefore saving significant computation time. Second, since the new loss calculation only requires forward-pass computation, it is quite natural to parallelize the computation into distributed machines to further accelerate the training.

Another point worth noting for the practical application of this method lies in the model capacity, which is highly related to the choice of the Gaussian noise level σ. We show that for large σ, the induced Gaussian smoothed models may not be expressive enough to approximate functions (i.e., learn solutions) with a large Lipschitz constant. Therefore, using a small value of σ is usually a better choice in practice. However, a small σ will lead to high-variance Stein's derivative estimation, which inevitably causes unstable training. We introduce several variance reduction methods that have been empirically verified to be effective for mitigating the problem. Further experiments demonstrate that, compared to standard PINN training, our proposed method can achieve competitive error but is significantly faster.

2 RELATED WORKS

2.1 Neural Approximation of PDE Solutions

Neural approximation approaches rely on governing equations and boundary conditions (or variants) to train neural networks to approximate the corresponding PDE solutions. Physics-Informed Neural Networks (PINN) [Sirignano and Spiliopoulos, 2018, Raissi et al., 2019] is one of the typical learning frameworks which constrains the output of deep neural networks to satisfy the given governing equations and boundary conditions. The application of PINN includes aerodynamic flows [Mao et al., 2020, Yang et al., 2019], power systems [Misyris et al., 2020], and nano optic [Chen et al., 2020]. Recently, there is also a growing body of works on studying the optimization and generalization properties of PINN. [Shin et al., 2020] proved that the learned PINN will converge to the solution under certain conditions. [Krishnapriyan et al., 2021] proposed to use curriculum regularization to avoid failures during PINN training. Different from the neural operator approaches [Lu et al., 2019, Li et al., 2020], the neural approximation methods can work in an unsupervised manner, without the need of labeled data generated by conventional PDE solvers.

2.2 Better Training for Physics-Informed Neural Networks

Despite the success of using PINN in solving various PDEs, researchers recently observed its training inefficiency in multiple aspects. For example, [Jagtap et al., 2020] discussed the architecture-wise inefficiency and introduced an adaptive activation function, which optimizes the network by dynamically changing the topology of the PINN loss function for different PDEs. The most relevant works related to our approach are [Sirignano and Spiliopoulos, 2018] and [Chiu et al., 2021], both of which tried to tackle the inefficiency in automatic differentiation by using numerical differentiation. In [Sirignano and Spiliopoulos, 2018], a Monte Carlo approximation method is proposed to estimate the numerical differentiation of second-order derivatives for some specific second-order PDEs. Concurrently to our work, [Chiu et al., 2021] introduced a method, which combines both auto-differentiation and numerical differentiation in PINN training to trade-off numerical truncation error and training efficiency. Compared to these two works, our designed approach can be applied to general PDEs and provide unbiased estimations of any derivatives without the need of back-propagation in the computation of the loss. Detailed discussions can be found in Section 4.3.

2.3 Gaussian Smoothed Model

Injecting Gaussian noise to the input has been popularly used in machine learning to improve model's robustness. [Li et al., 2019, Cohen et al., 2019] first used Gaussian smoothed models (a.k.a. randomized smoothing) to provide robustness guarantee when facing adversarial attacks. Since then, the smoothed models with different noise types have been developed for various scenarios [Zhai et al., 2020, Yang et al., 2020]. Leveraging Stein's Identity for efficient derivative estimation is not entirely new in machine learning. The method is one of the standard approaches in zero-order optimization where the exact derivatives cannot be obtained [Flaxman et al., 2004, Nesterov and Spokoiny, 2017, Liu et al., 2020, Pang et al., 2020]. To the best of our knowledge, there is no previous work using Stein's Identity on Gaussian smoothed models for efficient PINN training.

3 PRELIMINARY

Without loss of generality, we formulate any partial differential equation as:

$$\mathcal{L}u(x) = \phi(x), \quad x \in \Omega \subset \mathbb{R}^d \tag{1}$$

$$\mathcal{B}u(x) = 0, \quad x \in \partial\Omega, \tag{2}$$

where $\mathcal{L}$ is the partial differential operator and $\mathcal{B}$ is the boundary condition. We use x to denote the spatiotemporal-dependent variable, and use u as the solution of the problem.

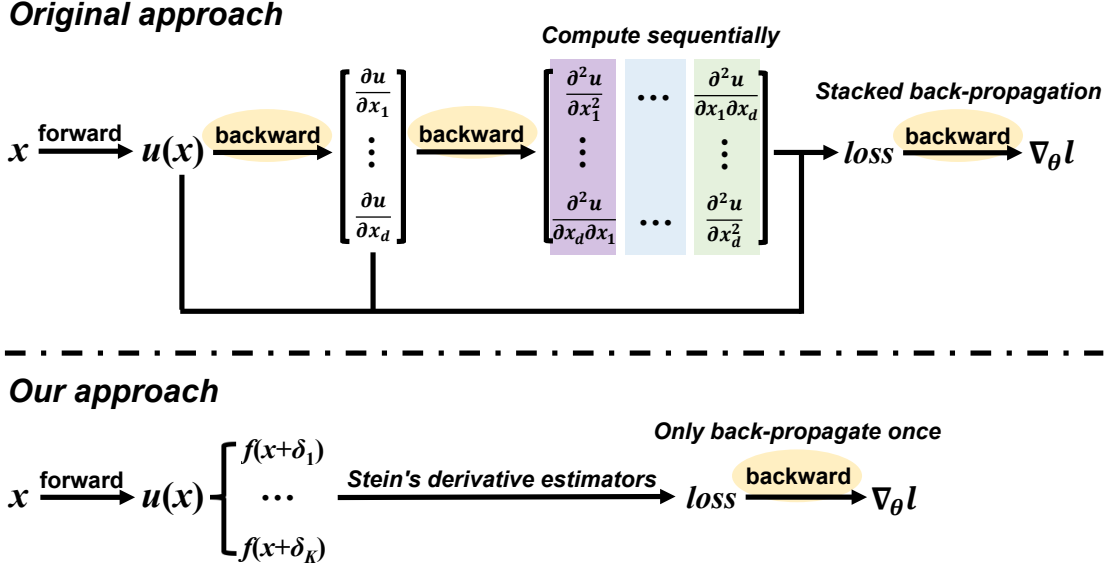**Original approach**

Our approach

Figure 1: **Training process of the original PINN approach and our approach**. The original approach requires multiple backward passes to compute the derivative terms in the loss. By contrast, our approach leverages the *Stein's derivative estimators* to compute the loss without back-propagation.

3.1 PINN Basics

Physics-informed Neural Network (PINN) [Raissi et al., 2019] is a popular choice to learn the function $u(x)$ automatically by minimizing the Physics-informed loss function induced by the governing equation (1) and boundary condition (2).

To be concrete, given any neural network $u(x;\theta)$ with parameter $\theta \in \Theta$, we define the Physics-informed loss function as

$$l_\Omega[\theta] = \|\mathcal{L}u(x;\theta) - \phi(x)\|^2_{L^2(\Omega)} \quad (3)$$

$$l_{\partial\Omega}[\theta] = \|\mathcal{B}u(x;\theta)\|^2_{L^2(\partial\Omega)}. \quad (4)$$

The loss term $l_\Omega[\theta]$ in Eqn. (3) corresponds to the PDE residual, which evaluates how $u(x;\theta)$ fits the partial differential equation, and $l_{\partial\Omega}[\theta]$ in Eqn. (4) corresponds to the boundary residual, which measures how well $u(x;\theta)$ satisfies the boundary condition. It is easy to see, if there exists θ^* that achieves zero loss in both residual terms $l_\Omega[\theta]$ and $l_{\partial\Omega}[\theta]$, then $u(x;\theta^*)$ will be a solution to the problem.

To find θ^* efficiently, PINN approaches leverage gradient-based optimization methods towards minimizing a linear combination of the two losses defined above. As the domain Ω and its boundary $\partial\Omega$ are usually continuous, Monte Carlo methods are used to approximate $l_\Omega[u]$ and $l_{\partial\Omega}[u]$ in practice. As a consequence, the optimization problem will be defined as

$$\arg\min_{\theta\in\Theta} \hat{l}_\Omega[\theta] + \lambda\hat{l}_{\partial\Omega}[\theta]. \quad (5)$$

In Eqn. (5), $\hat{l}_\Omega[\theta] = \frac{1}{N_1}\sum_{i=1}^{N_1}\|\mathcal{L}u(x^{(i)}) - \phi(x^{(i)})\|^2_2$ and $\hat{l}_{\partial\Omega}[\theta] = \frac{1}{N_2}\sum_{i=1}^{N_2}\|\mathcal{B}u(\tilde{x}^{(i)})\|^2_2$, where $\{x^{(1)},\cdots,x^{(N_1)}\}$

and $\{\tilde{x}^{(1)},\cdots,\tilde{x}^{(N_2)}\}$ are i.i.d sampled over Ω and $\partial\Omega$, N_1 and N_2 are the respective sample sizes, and λ is the coefficient used to balance the interplay between the two loss terms.

3.2 Inefficiency of PINN

It can be seen that the loss for PINN training, as shown in Eqn. (5), is defined on the network's derivatives. Therefore, the computation of this loss requires multiple back-propagation steps, which can be very inefficient, especially for high-dimensional high-order PDE problems. In details, there are two sources of inefficiency in computing the PINN loss. The first one lies in the order-level inefficiency due to the fact that different orders of derivatives can only be calculated sequentially: One has to first build up the computational graph for the first-order derivatives and then perform back-propagation on this graph to obtain the second-order derivatives. The second source is the dimension-level inefficiency which is a major issue for high-order PDE problems. In modern deep learning frameworks like `PyTorch` [Paszke et al., 2019], one has to perform back-propagation on the computational graph sequentially for each partial derivative to implement second-order operators like the Laplace operator, leading to a computational cost proportional to the dimensionality of the input [Pang et al., 2020, Meng et al., 2021].

Both of these two sources of inefficiency lead to a non-parallelizable training process, making the learning of PINN for high-dimensional high-order problems particularly slow. It can be easily obtained that for general k-order d-dimensional PDEs, the computational complexity

of each training iteration of PINN using back-propagation is $\mathcal{O}(d^{k-1})$. Therefore, it becomes substantially beneficial to explore new parallelizable methods for PINN training since it may largely increase the training efficiency by making full use of advanced computing hardware, such as GPU.

4 METHOD

In this work, we propose a novel method which enables fully parallelizable PINN training. The key of our approach is using a specific formulation of $u(x; \theta)$:

$$u(x; \theta) = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} f(x + \delta; \theta), \qquad (6)$$

where $f(\cdot; \theta)$ is a neural network with parameter θ, and σ is the noise level of the Gaussian distribution. When queried at x, u returns the expected output of f when its input is sampled from a Gaussian distribution $\mathcal{N}(x, \sigma^2 \boldsymbol{I})$ centered at x. It can be easily seen that $u(x; \theta)$ is a "smoothed" network constructed from a *"base" network* $f(x; \theta)$ by injecting Gaussian noise into the input, and we call $u(x; \theta)$ a *Gaussian smoothed model*. For simplicity, we may omit θ and refer to the base network and the Gaussian smoothed model as $f(x)$ and $u(x)$ in the rest part of the paper.

4.1 Back-propagation-free Derivative Estimators

At first glance, using formulation (6) brings more difficulties since the output of u can only be estimated by repeatedly sampling δ. However, we show that the efficiency can be significantly improved during training since all derivatives, as derived by Stein's Identity, can be calculated in parallel without using back-propagation.

Theorem 1 ([Stein, 1981]). *Suppose $x \in \mathbb{R}^d$. For any measurable function $f(x)$, define $u(x) = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} f(x + \delta)$, then we have*

$$\nabla_x u = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} [\frac{\delta}{\sigma^2} f(x + \delta)].$$

From the above theorem, we can see that the first-order derivative $\nabla_x u$ can be reformulated as an expectation term $\mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} [\frac{\delta}{\sigma^2} f(x + \delta)]$. To calculate the value of the expectation, we can use Monte Carlo method to obtain an unbiased estimation from K i.i.d Gaussian samples, i.e.,

$$\nabla_x u \approx \frac{1}{K} \sum_{k=1}^{K} \frac{\delta_k}{\sigma^2} f(x + \delta_k), \qquad (7)$$

where $\delta_k \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$, $k = 1, ..., K$.

It is easy to check that Stein's Identity can be extended to any order of derivatives by recursion. Here we showcase the corresponding identity for Hessian matrix and Laplace operator:

- Hessian matrix $\boldsymbol{H}u$:

$$\boldsymbol{H}u = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} \left[\left(\frac{\delta \delta^\top - \sigma^2 \boldsymbol{I}}{\sigma^4} \right) f(x + \delta) \right].$$

- Laplace operator Δu:

$$\Delta u = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} \left[\left(\frac{\|\delta\|^2 - \sigma^2 d}{\sigma^4} \right) f(x + \delta) \right].$$

For convenience, we refer to the Monte Carlo estimators based on these identities as *vanilla Stein's derivative estimators*.

We can plug the Stein's derivative estimators into the physics-informed loss functions (Eqn. 5) defined for a given PDE. In this way, we are able to overcome the two sources of inefficiency in training PINN mentioned earlier: We can see that Stein's derivative estimators for higher-order derivatives no longer require pre-computation for lower-order ones, and the derivatives with respect to each dimension can be obtained in a forward pass *simultaneously* instead of computing them dimension by dimension *sequentially*. Therefore, we can utilize the parallel computing power of GPU resource better and achieve $O(1)$ time complexity. See Figure 1 for an illustration of our approach.

These properties of Stein's derivative estimators are appealing because they enable a fully-parallelizable PINN training and lead to significant improvement over efficiency for solving high-dimensional PDEs. In the meantime, a natural concern about the deficient expressiveness of Gaussian smooth function may rise. In the following subsection, we will take deep discussion on this issue and demonstrate the importance of σ in practice to control the model expressiveness.

4.2 Model Capacity

In our method, Gaussian smoothed neural network is used instead of vanilla neural network as the solution of PDE. This modification brings a natural concern, i.e., whether the function space of the proposed Gaussian smoothed models is expressive enough to approximate the solutions of a given PDE problem. In this subsection, we show that the capacity of Gaussian smoothed neural networks is closely related to the Lipschitz function class according to the following theoretical result:

Theorem 2. *For any measurable function $f : \mathbb{R}^d \to \mathbb{R}$, define $u(x) = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})} f(x + \delta)$, then $u(x)$ is $\frac{F}{\sigma} \sqrt{\frac{2}{\pi}}$-Lipschitz with respect to ℓ_2-norm, where $F = \sup_{x \in \mathbb{R}^d} |f(x)|$.*

Proof. Theorem 1 states that

$$\nabla_x u = \mathbb{E} \left[\frac{\delta}{\sigma^2} f(x + \delta) \right].$$

Thus, for any unit vector α, we have

$$|\alpha^\top \nabla_x u| \le \frac{\mathbb{E}\left[|\alpha^\top \delta f(x+\delta)|\right]}{\sigma^2} \le \frac{F\mathbb{E}\left[|\alpha^\top \delta|\right]}{\sigma^2} = \frac{F}{\sigma}\sqrt{\frac{2}{\pi}}.$$

The last equality holds since $\alpha^\top \delta \sim \mathcal{N}(0, \sigma^2)$, which concludes the proof. □

Theorem 2 can be viewed as a negative result on the capacity of the Gaussian smoothed models. It indicates that the noise level σ is an important hyper-parameter that controls the expressive power of the model $u(x)$. For example, if the neural network f uses `tanh` activation in the final prediction layer, its output range will be restricted to $(-1, 1)$. From Theorem 2, it is straightforward to see that the Lipschitz constant of u is no more than $\frac{1}{\sigma}\sqrt{\frac{2}{\pi}}$ no matter how complex the network f is. In this setting, if we have a prior that the solution of a PDE has a large Lipschitz constant, we have to choose a small value of σ to approximate it well. On the other hand, using small σ would affect the finite-sample approximation error of vanilla Stein's derivative estimators. Without further assumptions on f, it is easy to check that the variance of vanilla Stein's derivative estimators in Eqn. (7) can be inversely proportional to σ. Thus, naively using Monte Carlo method requires a large sample size for small σ, which may even slow down the training in practice. In the next subsection, we present several variance reduction approaches that we find particularly useful during training.

4.3 Variance-Reduced Stein's Derivative Estimators

We mainly use two methods to reduce the variance for Stein's derivative estimators, the control variate method and the antithetic variable method. For simplicity, we demonstrate how to apply the two techniques to improve the estimator of $\nabla_x u$ and Δu, which can be easily extended to other Stein's derivative estimators.

The control variate method. One generic approach to reducing the variance of Monte Carlo estimates of integrals is to use an additive control variate [Evans and Swartz, 2000, Fishman, 2013, Hammersley, 2013], which is known as *baseline*. In our problem, we find $f(x)$ is a proper baseline which can lead to low-variance estimates of the derivative:

$$\nabla_x u = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 I)} \left[\frac{\delta}{\sigma^2}(f(x+\delta) - f(x)) \right]$$

$$\approx \frac{1}{K}\sum_{k=1}^{K}\left[\frac{\delta_k}{\sigma^2}(f(x+\delta_k) - f(x))\right]; \qquad (8)$$

$$\Delta u = \mathbb{E}\left[\left(\frac{\|\delta\|^2 - \sigma^2 d}{\sigma^4}\right)(f(x+\delta) - f(x))\right]$$

$$\approx \frac{1}{K}\sum_{k=1}^{K}\left[\left(\frac{\|\delta_k\|^2 - \sigma^2 d}{\sigma^4}\right)(f(x+\delta_k) - f(x))\right], \qquad (9)$$

where δ_k are i.i.d. samples from $\mathcal{N}(0, \sigma^2 I)$. To see clearly why this technique leads to variance reduction, we take Eqn. (8) as an example. we assume $f(x)$ is smooth leverage its Taylor expansion at x to rewrite $\frac{\delta}{\sigma^2}(f(x+\delta) - f(x))$ as $\frac{\delta}{\sigma^2}\left(\delta^\top \nabla f(x) + o(\|\delta\|)\right)$. This expression can be further simplified to $\xi\xi^\top\nabla f(x) + o(1)$, where $\xi = \delta/\sigma \sim \mathcal{N}(0, I)$. Therefore, the variance of the estimator in Eqn. (8) will be *independent* of σ. This fact is in sharp contrast to the original estimator provided in Eqn. (7). With a similar argument, we can also show that the variance of the estimator in Eqn. (9) is inversely of proportional to σ, while the the variance of the original estimator for Δu is inversely proportional to σ^2.

Further improvement using the antithetic variable method. The antithetic variable method is yet another powerful technique for variance reduction [Hammersley and Morton, 1956]. By using the symmetry of Gaussian distribution, it's easy to see that Eqn. (8) and (9) still holds when δ is substituted with $-\delta$, which leads to new estimators. Averaging the new estimator and the one in Eqn. (8) / (9) gives the following result:

$$\nabla_x u = \mathbb{E}\left[\frac{\delta}{2\sigma^2}(f(x+\delta) - f(x-\delta))\right]$$

$$\approx \frac{1}{K}\sum_{k=1}^{K}\left[\frac{\delta_k}{2\sigma^2}(f(x+\delta_k) - f(x-\delta_k))\right]; \quad (10)$$

$$\Delta u = \mathbb{E}\left[\left(\frac{\|\delta\|^2 - \sigma^2 d}{2\sigma^4}\right)(f(x+\delta) + f(x-\delta) - 2f(x))\right]$$

$$\approx \frac{1}{K}\sum_{k=1}^{K}\left[\left(\frac{\|\delta_k\|^2 - \sigma^2 d}{2\sigma^4}\right) \cdot \right.$$
$$\left. (f(x+\delta_k) + f(x-\delta_k) - 2f(x))\right], \qquad (11)$$

Again, by leveraging the Taylor expansion of $f(x)$, one can show that the variances of the estimators in Eqn. (10) and (11) are both *independent* of σ. For example, $\left(\frac{\|\delta\|^2 - \sigma^2 d}{2\sigma^4}\right)(f(x+\delta) + f(x-\delta) - 2f(x)) = \left(\frac{\|\delta\|^2 - \sigma^2 d}{2\sigma^4}\right)(\delta^\top H f(x)\delta + o(\|\delta\|^2))$, where $Hf(x)$ denotes the Hessian matrix of f at x. By letting $\xi = \delta/\sigma \sim \mathcal{N}(0, I)$, the estimator is simplified to $(\|\xi\|^2 - d)(\xi^\top H f(x)\xi + o(1))$. Thus, its variance is *independent* of σ. This property is especially appealing because it enables us to tune model expressiveness according to PDE complexity in practice. For instance, we can use a small σ to ensure the model is expressive enough to learn a complex PDE solution. We also provide empirical comparisons between vanilla Stein's derivative estimators and the improved ones. See Section 5.4 for details.

Note that Eqn. (8) and (11) look very similar to numerical differentiation in the surface form but they yield substantial differences. First, the roles of the term $f(x)$ are different.

In Eqn. (8) and (11), the term $f(x)$ is introduced as the baseline which doesn't change the value of the expectation since $\mathbb{E}\left[\frac{\delta}{\sigma^2}f(x)\right] = \mathbb{E}\left[\left(\frac{\|\delta\|^2-\sigma^2 d}{\sigma^4}\right)f(x)\right] = 0$. Therefore, multiplying any constant to $f(x)$ also holds, which will be infeasible for numerical differentiation. Second, our method provides *unbiased* estimation of the derivatives while numerical method can only obtain *biased* derivatives due to truncation error.

5 EXPERIMENTAL RESULTS

We conduct experiments to verify the effectiveness of our approach on a variety of PDE problems. Ablation studies on the design choices and hyper-parameters are then provided. Our codes are implemented based on `PyTorch` [Paszke et al., 2019]. All the models are trained on one NVIDIA GeForce RTX 2080 Ti GPU with 11GB memory and the reported training time is also measured on this machine. Our code can be found in the supplementary material.

5.1 Low-dimensional Problems

We first showcase our approach on two-dimensional PDE problems with visualization. In particular, we study the following two-dimensional Poisson's Equation with Dirichlet boundary conditions:

$$\begin{cases} \Delta u(x) = g(x) & x \in \Omega \\ u(x) = h(x) & x \in \partial\Omega \end{cases}$$

In our experiment, we set $\Omega = [0,1]^2$, $g(x) = -\sin(x_1 + x_2)$ and $h(x) = \frac{1}{2}\sin(x_1 + x_2)$. This PDE has a unique solution $u^*(x) = \frac{1}{2}\sin(x_1 + x_2)$.

We train a Gaussian-smoothed model to fit the solution with our method. Specifically, the base neural network of our model is a 4-layer MLP with 200 neurons and `tanh` activation in each hidden layer. In our Gaussian smoothed model, the noise level σ is set to 0.1, and the number of samples K is set to 2048. We use the variance-reduced derivative estimator in Eqn. (11) based on the control variate method and the antithetic variable method. To train the model, we use Adam as the optimizer [Kingma and Ba, 2015]. The learning rate is set to $3e-4$ in the beginning and then decays linearly to zero during training. In each training iteration, we sample $N_1 = 100$ points from the domain Ω and $N_2 = 100$ points from the boundary $\partial\Omega$ to obtain a mini-batch. We train the model for 1000 iterations. We use the variance-reduced derivative estimator in Eqn. (10) and (11) based on the control variate method and the antithetic variable method to compute the loss during training.

We compare our method with original PINN trained with stacked automatic differentiation [Raissi et al., 2019]. The PINN baseline model is trained using the same set of hyperparameters on the same machine to ensure a fair compari-

son. Evaluations are performed on a hold-out validation set which is unseen during training. We measure the accuracy of the learned solution by calculating the L^1 and L^2 relative error in the domain Ω. We also report the training time to study the efficiency in practice. The reported results are averaged across 3 different runs.

Experimental Results. The experimental results are summarized in Table 1. After training, our model reaches an L^1 relative error of 0.16%, indicating that the model fits the ground truth well. Furthermore, the final relative error of our model is very close to that of PINN, e.g., 0.13% v.s. 0.16% in terms of L^1 relative error. This result demonstrates that using Gaussian smoothed neural networks and Stein's derivative estimators does little harm to the accuracy of the model. Besides, one can notice that the training time of our model is slightly longer than that of PINN. We point out that this is because stacked back-propagation is especially time-consuming for *high dimensional problem*, while this experiment targets a two-dimensional PDE. In this case, removing the need for stacked back-propagation does not necessarily lead to speed-up, and this experiment is not a favorable setting for our method. That being said, this experiment still shows that using Gaussian smoothed neural networks and Stein's derivative estimators is feasible in PINN training.

We also examine the quality of the learned solution $u(x)$ by visualization. Figure 2 shows the ground truth $u^*(x)$, the learned solution $u(x)$ and the point-wise error $|u(x) - u^*(x)|$. We can see that the learned solution and the ground truth are very similar. Quantitatively speaking, the third subfigure in Figure 2 shows that the point-wise error of our model is less than $2e-3$ for most areas, thus the model can approximate the solution to Poisson's Equation with high accuracy when using our method.

5.2 High-dimensional Heat Equation

We consider the high-dimensional Heat Equation to demonstrate the speed and accuracy of our proposed method in solving high-dimensional PDE. Heat Equation is a prototypical parabolic PDE [Evans, 1998], which is deeply connected to several domains including probability theory [Lawler, 2010], image processing [Aubert et al., 2006], and quantum mechanics [Griffiths and Schroeter, 2018]. In this experiment, we study the following N-dimensional Heat Equation:

$$\begin{cases} u_t(x,t) = \Delta u(x,t) & x \in B(0,1), t \in (0,1) \\ u(x,0) = \|x\|^2/2N & x \in B(0,1) \\ u(x,t) = t + 1/2N & x \in \partial B(0,1), t \in [0,1] \end{cases}$$

In the above equation, $x \in \mathbb{R}^N$ is the spatial variable, while $t > 0$ is the temporal variable, which is slightly different from the notation in Eqn. (1) where x is denotes as the

Table 1: **Experimental results of solving Poisson's Equation (left) and HJB Equation (right).** "PINN" refers to the original PINN approach with stacked auto-differentiation. "Ours" refers to our new method using Gaussian smoothed models and Stein's derivative estimators. We report L^1 and L^2 relative error of the learned solutions, as well as the training time, to compare these two approaches.

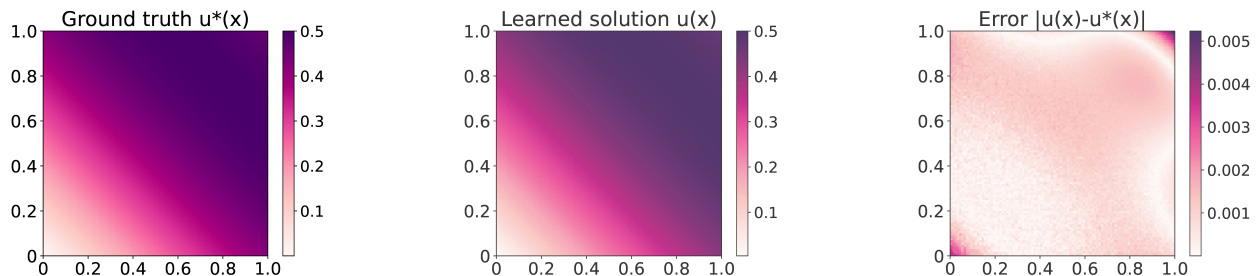| Problem	Method	L^1 error	L^2 error	Training time
2-dimensional Poisson's Equation	PINN	0.13%	0.15%	18.96s
	Ours	0.16%	0.19%	28.44s
100-dimensional Heat Equation	PINN	0.52%	0.60%	2.35min
	Ours	0.53%	0.63%	0.83min
250-dimensional HJB Equation	PINN + adv train	0.95%	1.18%	38.16h
	Ours + adv train	0.91%	1.37%	12.07h

Figure 2: **Visualization for two-dimensional Poisson's Equation**. Three subfigures show the ground truth $u^*(x)$, the learned solution $u(x)$ using our method, and the point-wise error $|u(x) - u^*(x)|$ respectively.

spatiotemporal-dependent variable in the PDE. We hope this abuse of notations does not confuse the readers. $B(0, 1)$ denotes the unit ball in N-dimensional space. Δ denotes the Laplacian operator with respect to the spatial variable x. The solution to the above equation is $u(x, t) = t + \|x\|^2/2N$. In our experiments, we focus on the high-dimensional setting and set the problem dimensionality N to 100.

Similar to Section 5.1, We compare our method with original PINN trained with stacked automatic differentiation [Raissi et al., 2019]. The baseline and our model share the same set of hyperparameters, and are trained on the same machine. Evaluations are performed on a hold-out validation set which is unseen during training. We report the accuracy of the learned solution by calculating the L^1 and L^2 relative error in the domain $B(0, 1) \times [0, 1]$, as well as the training time.

We train a Gaussian-smoothed model to fit the solution, where the base neural network of our model is a 4-layer MLP with 256 neurons and tanh activation in each hidden layer. In our Gaussian smoothed model, the noise level σ is set to 0.01, and the number of samples K is set to 2048. We use the variance-reduced derivative estimator in Eqn. (11) based on the control variate method and the antithetic variable method. The learning rate is set to $1e - 3$ in the beginning and then decays linearly to zero during training. Other training details can be found in the appendix.

Experimental Results. The experimental results are summarized in the middle of Table 1. Our model is as accurate as the PINN baseline. For example, our model and the original PINN obtain 0.53% and 0.52% L^1 relative error on the test set, respectively. Besides, our model is significantly more efficient than the PINN baseline, achieving $2.83\times$ acceleration in this problem. We emphasize that our model and the baseline are trained for *the same number of iterations*. Therefore, our technique clearly brings noticeable efficiency gains by avoiding stacked back-propagation.

5.3 High-dimensional Hamilton-Jacobi-Bellman Equation

We further use the high-dimensional Hamilton-Jacobi-Bellman (HJB) Equation to showcase the effectiveness of our proposed method in solving complicated non-linear high-dimensional PDE. HJB Equation is one of the most important non-linear PDE in optimal control theory. It has a wide range of applications, including physics [Sieniutycz, 2000], biology [Li et al., 2011], and finance [Pham, 2009]. Its discrete-time counterpart is the Bellman Equation widely used in reinforcement learning [Sutton and Barto, 2018].

Experimental Design. Following existing works [Han et al., 2018, Wang et al., 2022], we study the classi-

cal linear-quadratic Gaussian (LQG) control problem in N dimensions, whose HJB Equation is a second-order PDE[1] defined as below:

$$\begin{cases} u_t + \Delta u - \mu \|\nabla_x u\|^2 = 0 & x \in \mathbb{R}^N, t \in [0, T] \\ u(T, x) = g(x) & x \in \mathbb{R}^N \end{cases}$$

(12)

As is shown in [Han et al., 2018], there is a unique solution to Eqn. (12):

$$u(x, t) = -\frac{1}{\mu(2\pi)^{\frac{n}{2}}} \ln \int_{\mathbb{R}^n} e^{-\frac{\|y\|^2}{2} - \mu g(x - \sqrt{2(T-t)}y)} dy.$$

We set the parameters $\mu = 1$, $T = 1$, and the terminal cost function $g(x) = \ln \left(\frac{1 + \|x\|^2}{2} \right)$. To evaluate the training speed and performance in high dimensional cases, we set the problem dimensionality N to 250.

We compare our method with the sate-of-the-art PINN-based approach on HJB Equation [Wang et al., 2022]. Specifically, [Wang et al., 2022] shows that original PINN training algorithm cannot guarantee to learn an accurate solution to a large class of HJB Equation, and propose to use adversarial training for PINN to learn the solution with theoretical guarantee. The resulting training algorithm is powerful yet time-consuming because it involves both stacked back-propagation for high-dimensional function and additional computation over-head in adversarial training. In this experiment, we follow [Wang et al., 2022] to apply adversarial training, while removing stacked back-propagation with our method.

We train a Gaussian-smoothed model to fit the solution, where the base neural network of our model is a 4-layer MLP with 768 neurons and $\tanh$ activation in each hidden layer. The noise level σ, the number of samples K, and the derivative estimators are the same as those in Section 5.2. The learning rate is set to $2e - 4$ in the beginning and then decays linearly to zero during training. Other training settings, including the batch size, the total iterations, etc., are the same as those in [Wang et al., 2022], and the details can be found in the appendix.

Experimental Results. The experimental results are summarized in the bottom of Table 1, where the performance of the baseline "PINN + adv train" is reported in [Wang et al., 2022]. It's clear that the accuracy of our model is on par with the state-of-the-art result, showing that using the Gaussian smoothed model and approximated derivatives does not hurt the model performance. Besides, our model and the baseline are trained for the same number of iterations, and the reported training time indicates that our method is much more efficient compared with stacked back-propagation in high-dimensional problem. To be specific,

---

[1]Similar to the Heat Equation in Section 5.2, x denotes the state variable and t denotes the temporal-dependent variable here.

our method is $3.16\times$ faster than the baseline, which largely reduces the training cost.

These observations clearly demonstrates that our method can significantly accelerate the training for high-dimensional PDE problems without sacrificing performance. We believe this is an initial step towards efficiently solving high-dimensional PDEs using deep learning approaches.

5.4 Ablation studies

we conduct ablation studies in this section to examine the effects of different design choices.

Regarding the choice of derivative estimators. To compare different statistical estimators introduced in the paper, we conduct several numerical experiments and study the approximation errors of each estimator. We experiment with a Gaussian-smoothed model $u(x) = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 I)} f(x + \delta)$, where the base neural network $f(x)$ is a randomly initialized and then fixed 4-layer MLP. The input/output dimension is set to 1000/1 respectively. We set the noise level σ to 0.1. We consider the derivative estimators for the gradient $\nabla_x u$ and the Laplace operator Δu. We randomly sample 10^3 points in $[0, 1]^{1000}$. For each sampled point x, we sample 10^5 Gaussian noises and use back-propagation to calculate $\nabla_x u$ and Δu as an oracle. Empirically, we observe the output of the oracle is very stable, whose variance is less than $1e - 7$. We compute the L_1 distance between the derivatives returned by the oracle and the derivative estimators used in the paper as the evaluation metric. We compare three estimators: a) the vanilla Stein's derivative estimators, defined in Eqn. (7); b) variance-reduced estimators with the control variate method, defined in Eqn. (8) and (9); c) variance-reduced estimators with both the control variate and the antithetic variable method, defined in Eqn. (10) and (11). For each estimator, we vary the sample size K from 8 to 32768.

The results are shown in Figure 3. For the first-order derivative, i.e., $\nabla_x u$, the vanilla Stein's derivative estimator is already accurate, and the variance-reduced ones further improve it slightly. For the second-order derivative, i.e., Δu, the vanilla Stein's derivative estimator performs poorly, whose error is larger than $1e - 2$ even with sample size $K = 32768$. The control variate method slightly improves, but the resulting estimator is still inaccurate given reasonable sample size. Combining the control variate method with the antithetic variable method significantly reduces the approximation error: The corresponding estimator is $100\times$ more accurate than the vanilla one. These observations suggest that variance reduction is essential in using Stein's Identity, especially for estimating high-order derivatives.

Regarding the choice of the noise level σ. As discussed in Section 4.2, the noise level σ controls the expressive power of the Gaussian-smoothed model $u(x)$, which will greatly
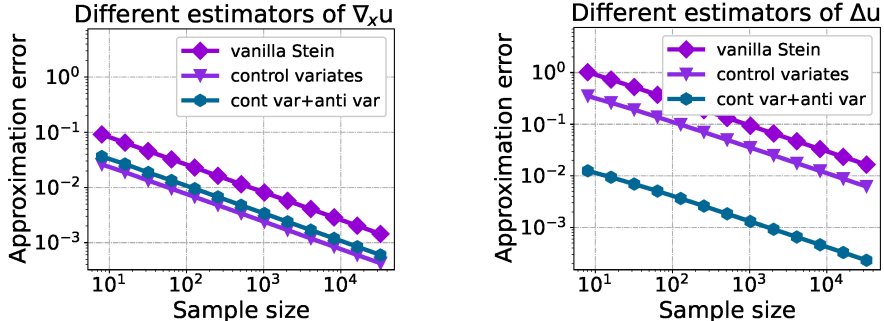
Figure 3: **Numerical results on the approximation error**. The two subfigures show the approximation error of different estimators for the gradient (left) and the Laplace operator (right), where the x-axis indicates the logarithmic-scaled sample sizes and the y-axis indicates the logarithmic-scaled approximation error.

Table 2: **Experimental results for ablation studies.** We experiment on the 250-dimensional HJB Equation using different hyperparameters. The left table investigates the impact of the noise level σ on the final performance. The right table compare the performances of our model trained with different sample size K in the Stein's derivative estimator.

Noise level σ	L^1 error	L^2 error
1	22.74%	27.90%
$1e-1$	1.04%	1.42%
$1e-2$	0.91%	1.37%
$1e-3$	0.96%	1.39%

Sample size K	L^1 error	L^2 error	Training time
256	1.11%	1.42%	1.34h
512	0.98%	1.32%	2.63h
1024	0.96%	1.38%	5.59h
2048	0.91%	1.37%	12.07h

affect the performance of the learned models. To understand the influence of σ during training, we conduct experiments on 250-dimensional HJB Equation with σ ranging from $1e-3$ to 1. In all these experiments, we set the sample size K to 2048. We tune the learning rate and the loss coefficient in the experiment and report the *best* result for each setting.

The results are shown in the left part of Table 2. Note that changing the noise level σ does not affect the computational complexity. Thus, the training time of different models are nearly the same, and we only report the relative error for brevity. It can be seen that the learned solutions are accurate when $\sigma \leq 0.1$, and the value of σ does not affect the model's performance much. However, when σ becomes too large, e.g., $\sigma = 1$, the relative error of the model is high at the end of training. This indicates that the solution of the PDE may not be in the function class that the model can express. Note that σ cannot be arbitrarily small because the calculation of the second-order derivatives involves small values, e.g., σ^4, which will introduce round-off error when we use float precision.

Regarding the choice of sample size K. In our method, the sample size K is a hyper-parameter that controls the speed-accuracy trade-off. When K is larger, the approximation of the derivatives is more accurate, while the training would be slower. We run experiments on 250-dimensional HJB Equation with the sample size K ranging from 256 to 2048. In all these experiments, we set the noise level σ to 0.01. We tune the learning rate and the loss coefficient in the

experiment and report the *best* result for each setting.

The results are shown in the right part of Table 2. From the results we can see that a larger sample size K can lead to better performance, at the cost of increased training time. However, the training is not very sensitive to the sample size. Even with only 256 samples, our method is still accurate (and extremely efficient). This is because the variance-reduced Stein's derivative estimators can provide sufficiently accurate estimation given a small sample size.

6 CONCLUSION

In this paper, we develop a novel approach that can significantly accelerate the training of Physics-Informed Neural Networks. In particular, we parameterize the PDE solution by the Gaussian smoothed model and show that, as derived from Stein's Identity, the second-order derivatives can be efficiently calculated without back-propagation. Experimental results show that our proposed method can significantly accelerate PINN without sacrificing accuracy. One limitation of our work is that our method only leads to acceleration for high dimensional PDE. Accelerating PINN on more classes of problems can be an exciting direction for future work. We believe this work is an initial step towards efficiently solving the high-dimensional partial differential equations using deep learning approaches and will address various other challenges along the way.

References

[Aubert et al., 2006] Aubert, G., Kornprobst, P., and Aubert, G. (2006). *Mathematical problems in image processing: partial differential equations and the calculus of variations*, volume 147. Springer.

[Chen et al., 2020] Chen, Y., Lu, L., Karniadakis, G. E., and Dal Negro, L. (2020). Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics express*, 28(8):11618–11633.

[Chiu et al., 2021] Chiu, P.-H., Wong, J. C., Ooi, C., Dao, M. H., and Ong, Y.-S. (2021). Can-pinn: A fast physics-informed neural network based on coupled-automatic-numerical differentiation method.

[Cohen et al., 2019] Cohen, J., Rosenfeld, E., and Kolter, Z. (2019). Certified adversarial robustness via randomized smoothing. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320, Long Beach, California, USA. PMLR.

[Evans, 1998] Evans, L. C. (1998). Partial differential equations. *Graduate studies in mathematics*, 19(4):7.

[Evans and Swartz, 2000] Evans, M. and Swartz, T. (2000). *Approximating integrals via Monte Carlo and deterministic methods*, volume 20. OUP Oxford.

[Fishman, 2013] Fishman, G. (2013). *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media.

[Flaxman et al., 2004] Flaxman, A. D., Kalai, A. T., and McMahan, H. B. (2004). Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*.

[Griffiths and Schroeter, 2018] Griffiths, D. J. and Schroeter, D. F. (2018). *Introduction to quantum mechanics*. Cambridge university press.

[Hammersley, 2013] Hammersley, J. (2013). *Monte carlo methods*. Springer Science & Business Media.

[Hammersley and Morton, 1956] Hammersley, J. and Morton, K. (1956). A new monte carlo technique: antithetic variates. In *Mathematical proceedings of the Cambridge philosophical society*, volume 52, pages 449–475. Cambridge University Press.

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

[Jagtap et al., 2020] Jagtap, A. D., Kawaguchi, K., and Karniadakis, G. E. (2020). Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136.

[Khoo et al., 2017] Khoo, Y., Lu, J., and Ying, L. (2017). Solving parametric pde problems with artificial neural networks. *arXiv preprint arXiv:1707.03351*.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*.

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

[Lawler, 2010] Lawler, G. F. (2010). *Random walk and the heat equation*, volume 55. American Mathematical Soc.

[Li et al., 2019] Li, B., Chen, C., Wang, W., and Carin, L. (2019). Second-order adversarial attack and certifiable robustness.

[Li et al., 2011] Li, W., Todorov, E., and Liu, D. (2011). Inverse optimality design for biological movement systems. *IFAC Proceedings Volumes*, 44(1):9662–9667.

[Li et al., 2020] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2020). Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.

[Liu et al., 2020] Liu, S., Chen, P.-Y., Kailkhura, B., Zhang, G., Hero III, A. O., and Varshney, P. K. (2020). A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5):43–54.

[Long et al., 2019] Long, Z., Lu, Y., and Dong, B. (2019). Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925.

[Long et al., 2018] Long, Z., Lu, Y., Ma, X., and Dong, B. (2018). Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR.

[Lu et al., 2019] Lu, L., Jin, P., and Karniadakis, G. E. (2019). Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*.

[Mao et al., 2020] Mao, Z., Jagtap, A. D., and Karniadakis, G. E. (2020). Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789.

[Meng et al., 2021] Meng, C., Song, Y., Li, W., and Ermon, S. (2021). Estimating high order gradients of the data distribution by denoising. *Advances in Neural Information Processing Systems*, 34.

[Misyris et al., 2020] Misyris, G. S., Venzke, A., and Chatzivasileiadis, S. (2020). Physics-informed neural networks for power systems. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE.

[Nesterov and Spokoiny, 2017] Nesterov, Y. and Spokoiny, V. (2017). Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566.

[Pang et al., 2020] Pang, T., Xu, K., Li, C., Song, Y., Ermon, S., and Zhu, J. (2020). Efficient learning of generative models via finite-difference score matching. *Advances in Neural Information Processing Systems*, 33:19175–19188.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037.

[Pham, 2009] Pham, H. (2009). *Continuous-time stochastic control and optimization with financial applications*, volume 61. Springer Science & Business Media.

[Raissi et al., 2019] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). 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.

[Shin et al., 2020] Shin, Y., Darbon, J., and Karniadakis, G. E. (2020). On the convergence and generalization of physics informed neural networks. *arXiv e-prints*, pages arXiv–2004.

[Sieniutycz, 2000] Sieniutycz, S. (2000). Hamilton–jacobi–bellman framework for optimal control in multistage energy systems. *Physics Reports*, 326(4):165–258.

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

[Stein, 1981] Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Wang et al., 2022] Wang, C., Li, S., He, D., and Wang, L. (2022). Is L^2 physics-informed loss always suitable for training physics-informed neural network? In *Advances in Neural Information Processing Systems*.

[Yang et al., 2020] Yang, G., Duan, T., Hu, J. E., Salman, H., Razenshteyn, I., and Li, J. (2020). Randomized smoothing of all shapes and sizes. In *International Conference on Machine Learning*, pages 10693–10705. PMLR.

[Yang et al., 2019] Yang, X., Zafar, S., Wang, J.-X., and Xiao, H. (2019). Predictive large-eddy-simulation wall modeling via physics-informed neural networks. *Physical Review Fluids*, 4(3):034602.

[Zhai et al., 2020] Zhai, R., Dan, C., He, D., Zhang, H., Gong, B., Ravikumar, P., Hsieh, C.-J., and Wang, L. (2020). Macer: Attack-free and scalable robust training via maximizing certified radius. *arXiv preprint arXiv:2001.02378*.

A   OMITTED PROOFS

In this section, we always assume $x \in \mathbb{R}^d$, $f(x)$ is a measurable function, and that $u(x) = \mathbb{E}_{\delta \sim \mathcal{N}(0,\sigma^2 I)} f(x + \delta)$.

A.1   Proof of Theorem 1

Proof. Note that

$$u(x) = (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} e^{-\frac{\|\delta\|^2}{2\sigma^2}} f(x + \delta) \mathrm{d}\delta = (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} e^{-\frac{\|t-x\|^2}{2\sigma^2}} f(t) \mathrm{d}t. \tag{13}$$

We have

$$\nabla_x u(x) = (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} \nabla_x e^{-\frac{\|t-x\|^2}{2\sigma^2}} f(t) \mathrm{d}t = (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} \frac{t-x}{\sigma^2} e^{-\frac{\|t-x\|^2}{2\sigma^2}} f(t) \mathrm{d}t$$

$$= (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} \frac{\delta}{\sigma^2} e^{-\frac{\|\delta\|^2}{2\sigma^2}} f(x + \delta) \mathrm{d}\delta = \mathbb{E}_{\delta \sim \mathcal{N}(0,\sigma^2 I)} \left[\frac{\delta}{\sigma^2} f(x + \delta) \right],$$

which concludes the proof. $\square$

A.2   Deviations of the Second Order Stein's derivative estimators

Proposition 3. *We have*

$$\boldsymbol{H}u = \mathbb{E}_{\delta \sim \mathcal{N}(0,\sigma^2 I)} \left[\left(\frac{\delta\delta^\top - \sigma^2 \boldsymbol{I}}{\sigma^4} \right) f(x + \delta) \right]; \tag{14}$$

$$\Delta u = \mathbb{E}_{\delta \sim \mathcal{N}(0,\sigma^2 I)} \left[\left(\frac{\|\delta\|^2 - \sigma^2 d}{\sigma^4} \right) f(x + \delta) \right]. \tag{15}$$

Proof. Note that $u(x)$ can be expressed as in Eqn.(13). Therefore, for $i, j \in \{1, \cdots, d\}$ and $i \neq j$, we have

$$\frac{\partial^2}{\partial x_i \partial x_j} u(x) = (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} \frac{\partial^2}{\partial x_i \partial x_j} e^{-\frac{\|t-x\|^2}{2\sigma^2}} f(t) \mathrm{d}t = (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} \frac{(t_i - x_i)(t_j - x_j)}{\sigma^4} e^{-\frac{\|t-x\|^2}{2\sigma^2}} f(t) \mathrm{d}t$$

$$= (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} \frac{\delta_i \delta_j}{\sigma^4} e^{-\frac{\|\delta\|^2}{2\sigma^2}} f(x + \delta) \mathrm{d}\delta = \mathbb{E}_{\delta \sim \mathcal{N}(0,\sigma^2 I)} \left[\frac{\delta_i \delta_j}{\sigma^4} f(x + \delta) \right], \tag{16}$$

For the case where $i = j$, similar computations will yield

$$\frac{\partial^2}{\partial x_i^2} u(x) = (2\pi)^{-\frac{n}{2}} \int_{\mathbb{R}^d} \frac{\partial^2}{\partial x_i^2} e^{-\frac{\|t-x\|^2}{2\sigma^2}} f(t) \mathrm{d}t = \mathbb{E}_{\delta \sim \mathcal{N}(0,\sigma^2 I)} \left[\frac{\delta_i^2 - 1}{\sigma^4} f(x + \delta) \right], \tag{17}$$

By definition of Hessian matrix, Eqn.(14) can be proven by combining Eqn.(16) and (17).

As for Eqn.(15), note that $\Delta u = \mathrm{tr}(\boldsymbol{H}u)$, where $\mathrm{tr}(\cdot)$ denotes the trace of a matrix. Therefore,

$$\Delta u = \mathbb{E} \left[\mathrm{tr} \left(\frac{\delta\delta^\top - \sigma^2 \boldsymbol{I}}{\sigma^4} \right) f(x + \delta) \right] = \mathbb{E} \left[\left(\frac{\|\delta\|^2 - \sigma^2 d}{\sigma^4} \right) f(x + \delta) \right],$$

which concludes the proof. $\square$

A.3   Deviations of the Variance-Reduced Stein's derivative estimators

Proposition 4 (Estimators based on the control variate method)**.**

$$\nabla_x u = \mathbb{E}_{\delta \sim \mathcal{N}(0,\sigma^2 I)} \left[\frac{\delta}{\sigma^2} (f(x + \delta) - f(x)) \right]; \tag{18}$$

$$\Delta u = \mathbb{E} \left[\left(\frac{\|\delta\|^2 - \sigma^2 d}{\sigma^4} \right) (f(x + \delta) - f(x)) \right]. \tag{19}$$

Proof. Note that $\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$ implies $\mathbb{E}[\delta] = 0$ and $\mathbb{E}[\|\delta\|^2 - \sigma^2 d] = 0$.

Thus, $\mathbb{E}[\delta f(x)] = \mathbb{E}[\|\delta\|^2 - \sigma^2 df(x)] = 0$. Applying Theorem 1 and Proposition 3, Eqn.(18) and (19) hold by linearity of expectation. □

Proposition 5 (Estimators based on the control variate and antithetic variable method)**.**

$$\nabla_x u = \mathbb{E}\left[\frac{\delta}{2\sigma^2}(f(x+\delta) - f(x-\delta))\right]; \tag{20}$$

$$\Delta u = \mathbb{E}\left[\left(\frac{\|\delta\|^2 - \sigma^2 d}{2\sigma^4}\right)(f(x+\delta) + f(x-\delta) - 2f(x))\right] \tag{21}$$

Proof. Note that $\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$ implies $-\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$. Thus, we can substitute δ with $-\delta$ in Eqn.(18) and obtain

$$\nabla_x u = \mathbb{E}_{\delta \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})}\left[-\frac{\delta}{\sigma^2}(f(x-\delta) - f(x))\right] \tag{22}$$

The summation of Eqn.(18) and (22) gives Eqn.(20). Eqn.(21) is proven similarly. □

B   EXPERIMENTAL SETTINGS

B.1   Poisson's Equation

Hyperparameters.   The hyperparameters used in our experiment on Poisson's Equation are described in Table 3.

Table 3: **Derailed experimental settings** of Poisson's Equation.

Model Configuration	
Layers	4
Hidden dimension	256
Activation	tanh
Noise level σ	0.01
Sample size K	2048
Hyperparameters	
Total iterations	1000
Domain Batch Size N_1	100
Boundary Batch Size N_2	100
Boundary Loss Weight λ	300
Learning Rate	$1e-3$
Learning Rate Decay	Linear
Adam ε	$1e-8$
Adam(β_1, β_2)	$(0.9, 0.999)$

Training data.   The training data is sampled *online*. Specifically, in each iteration, we sample N_1 i.i.d. data points, $x^{(1)}, \cdots, x^{(N_1)}$, uniformly from the domain Ω, and N_2 i.i.d. data points, $\tilde{x}^{(1)}, \cdots, \tilde{x}^{(N_2)}$, uniformly from the boundary $\partial\Omega$.

B.2   Heat Equation

Hyperparameters.   The hyperparameters used in our experiment on HJB Equation are described in Table 4.

Training data.   The training data is sampled *online*. Specifically, in each iteration, we *uniformly* sample N_1 i.i.d. data points, $(x^{(1)}, t^{(1)}), \cdots, (x^{(N_1)}, t^{(N_1)})$, from the domain $B(0, 1) \times (0, 1)$; N_2 i.i.d. data points, $(\tilde{x}^{(1)}, 0), \cdots, (\tilde{x}^{(N_2)}, 0)$, from $B(0, 1) \times \{0\}$; and N_3 i.i.d. data points, $(\hat{x}^{(1)}, \hat{t}^{(1)}), \cdots, (\hat{x}^{(N_3)}, \hat{t}^{(N_3)})$, from $\partial B(0, 1) \times [0, 1]$.

Table 4: **Derailed experimental settings** of Heat Equation.

Model Configuration	
Layers	4
Hidden dimension	256
Activation	tanh
Noise level σ	0.01
Sample size K	2048
Hyperparameters	
Total iterations	1000
Domain Batch Size N_1	50
Initial Condition Batch Size N_2	50
Spatial Boundary Batch Size N_3	50
Initial Condition Weight λ_2	1000
Spatial Boundary Weight λ_3	1000
Learning Rate	$1e-3$
Learning Rate Decay	Linear
Adam ε	$1e-8$
Adam(β_1**,** β_2**)**	(0.9, 0.999)

B.3 HJB Equation

Hyperparameters.   The hyperparameters used in our experiment on HJB Equation are described in Table 5.

Table 5: **Derailed experimental settings** of HJB Equation.

Model Configuration	
Layers	4
Hidden dimension	768
Activation	tanh
Noise level σ	0.01
Sample size K	2048
Hyperparameters	
Total iterations	10000
Domain Batch Size N_1	50
Boundary Batch Size N_2	50
Boundary Loss Weight λ	500
(Adversarial training) Inner Loop Iterations K	20
(Adversarial training) Inner Loop Step Size η	0.05
Learning Rate	$2e-4$
Learning Rate Decay	Linear
Adam ε	$1e-8$
Adam(β_1**,** β_2**)**	(0.9, 0.999)

Training data.   The training data is sampled *online*. Specifically, in each iteration, we sample N_1 i.i.d. data points, $(x^{(1)}, t^{(1)}), \cdots, (x^{(N_1)}, t^{(N_1)})$, from the domain $\mathbb{R}^n \times [0, T]$, and N_2 i.i.d. data points, $(\tilde{x}^{(1)}, T), \cdots, (\tilde{x}^{(N_2)}, T)$, from the boundary $\mathbb{R}^n \times \{T\}$, where $(x^{(i)}, t^{(i)}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_n) \times \mathcal{U}(0, 1)$ and $\tilde{x}^{(j)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_n)$.