PINNs with Learnable Quadrature

Sourav Pal
UW-Madison
spal9@wisc.edu

Kamyar Azizzadenesheli
NVIDIA Corporation
kaazizzad@gmail.com

Vikas Singh
UW-Madison
vsingh@biostat.wisc.edu

Abstract

The growing body of work on Physics-Informed Neural Networks (PINNs) seeks to use machine learning strategies to improve methods for solution discovery of Partial Differential Equations (PDEs). While classical solvers may remain the preferred tool of choice in the short-term, PINNs can be viewed as complementary. The expectation is that in some specific use cases, they can be effective, standalone. A key step in training PINNs is selecting domain points for loss evaluation, where ***Monte Carlo*** sampling remains the dominant but often suboptimal in low dimension settings, common in physics. We leverage recent advances in ***asymptotic expansions*** of quadrature nodes and weights (for weight functions belonging to the modified Gauss-Jacobi family) together with suitable adjustments for parameterization towards a data-driven framework for learnable quadrature rules. A direct benefit is a performance improvement of PINNs, relative to existing alternatives, on a wide range of problems studied in the literature. Beyond finding a standard solution for an instance of a single PDE, our construction enables learning rules to predict solutions for a given family of PDEs via hyper-networks, a useful capability for PINNs.

1 Introduction

Differential equations are widely used across science, providing a framework for modeling/analyzing diverse physical dynamics. Most real-world settings lead to differential equations where analytical solutions are not possible, but research over decades has led to a mature set of numerical methods [Ames, 2014, Trefethen and Bau, 2022]. Specifically, a growing body of work has identified novel architectures, by marrying differential equation solvers with deep learning and these formulations offer new capabilities. For example, one now has access to completely data-driven approaches [Li et al., 2020b,a, Kovachki et al., 2021] which use observational data to estimate the operator for a PDE. We also have a new class of differential equation solvers that exploit neural networks to encode physical laws (PINNs) [Raissi et al., 2019, Kharazmi et al., 2019] without any observational data. PINNs give solutions which are mesh-independent, easier to implement due to automatic differentiation, and handle non-linearity well. As such, while classical solvers may remain the default choice in general, developments in PINN suggest that they can be an alternative in certain scenarios.

Roughly speaking, the aforementioned line of work [Karniadakis et al., 2021], discussed in more detail later in §7 can be broadly classified under three main threads: (a) Solving PDEs using neural networks (PINNs) [Raissi et al., 2019]; (b) PDE discovery (e.g., symbolic regression) [Holt et al., 2023, d'Ascoli et al., 2023] and (c) operator learning (e.g., Fourier Neural Operator) [Azizzadenesheli et al., 2024]. This classification is loosely based on the amount of data or physics used to solve/inform the forward/inverse problem, [Boullé and Townsend, 2023]. Our work falls under the first category, where we wish to solve a single (or multiple instances) of a PDE with PINNs. Here, we seek to identify how a learning mechanism can deliver efficiency gains solely based on the shared structure and knowledge of physics, *without the use of any labeled data*.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

PDEs and Quadrature. Consider the second-order PDE,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \log(x)\sin(y) + f(x,y)y^3 \tag{1}$$

One way to find u is to integrate both sides with a test function $v(x,y)$ resulting in:

$$\int\int \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) v(x,y)\mathrm{d}x\mathrm{d}y = \int\int \left(\log(x)\sin(y) + f(x,y)y^3\right) v(x,y)\mathrm{d}x\mathrm{d}y \tag{2}$$

Doing so enables the use of numerical methods which build-up sums that converge to the integral's true value. This has the benefit of easing regularity conditions on u. It solves the original problem in a weighted sense. A *quadrature method* [Golub and Welsch, 1969] chooses evaluation points (nodes) and corresponding weights to minimize approximation error. These points/nodes may be constant step/uniform or adaptive, i.e., either fixed or adaptive *quadrature rules* for estimating the integral.

Challenges in computing an integral. In many applications from fluid dynamics (turbulent flow) [Kutz, 2017] to radiation treatment planning (fluence calculation at tissue interfaces) [Lou et al., 2021, Beckham et al., 2002] to materials science (fracture mechanics) [Aliabadi and Rooke, 1991, Rice and Tracey, 1973], the associated data involves irregular behavior including singularities. Uniformly splitting the domain of integration into equal sub-domains, in many cases, is insufficient owing to the singularity associated with the integrand. Further, even a sophisticated partitioning scheme, runs into difficulties in the multi-dimensional case. A common solution is to use some variant of Monte Carlo sampling. In higher dimensions, we have no choice but to sample at large and expect the estimated solution to converge to the true solution, given enough runtime. In lower dimensions, Monte Carlo sampling is sub-optimal [Lu et al., 2021] and several strategies to improve the speed and accuracy of the integral computation are known, the prominent ones being some variant of **adaptive quadrature scheme**, essentially choosing an adaptive grid dependent on the integrand.

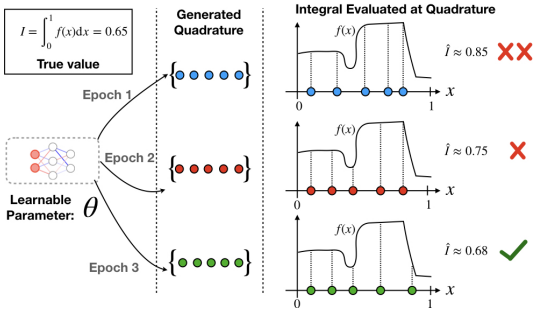


Figure 1: Relevance of learnable quadratures. Given a fixed number of quadrature points, one can update a learnable module based on how good/bad the numerical approximation of the integral is.

Main Idea. We propose a learnable quadrature scheme (Fig.1) utilizing a rich theory of *orthogonal polynomials* (OP) and *asymptotic expansions*. Consider solving a PDE with PINNs either in its *strong form* (PINN) [Karniadakis et al., 2021] or *weak form* (VPINN, hp-VPINN) [Kharazmi et al., 2019, 2021]. In both cases, the goal is to either (a) determine *which points in the domain* to evaluate the function on (this is the *strong form*) or (b) determine *which test functions to use* (for evaluation of the *weak form*). We tie these choices to the roots of OP w.r.t. the modified *Gauss-Jacobi* weight functions. Next, we use recent advances in asymptotic expansions of quadrature nodes and weights [Opsomer and Huybrechs, 2023] to achieve a scheme to compute these *efficiently*.

Contributions: For solving PDE with PINNs, we describe two separate learning modules. *First* is the solution function for the PDE parameterized using a neural network (standard in PINNs). The *second* is a parameterized weight function which induces a family of OP. Our parameterization of the weight function together with asymptotic expansions takes advantage of *parallel* compute on GPUs to generate a massive number *(millions)* of quadrature nodes and weights in near *constant* time. This provides an alternative to Monte Carlo sampling for *low-dimensional* problems. Our model can solve most PDEs from the PINN literature achieving better performance than existing adaptive/non-adaptive sampling schemes. Our parameterization of the weight function also allows learning a quadrature predictor for a family of PDEs (with shared structure), which may be of independent interest.

2 Preliminaries

We review some key concepts that will be useful throughout.

Orthogonal Polynomials (OP). A sequence of real-valued polynomials $p_0(x), p_1(x), p_2(x), ...,$ where each $p_n(x)$ is a polynomial of degree n are *orthogonal* [Olver et al., 2020] with respect to a

continuous and non-negative weight function $w(x)$ defined in the interval (a, b) if:

$$\langle p_m, p_n \rangle_w = \int_a^b p_m(x) p_n(x) w(x) \mathrm{d}x = \begin{cases} 0 & \text{if } m \neq n, \\ h_n & \text{if } m = n \end{cases} \tag{3}$$

where h_n is a normalization constant. In fact, if $h_n = 1$ for all n, then the family is orthonormal.

Modified Gauss-Jacobi Weight functions. We use weight functions from the modified Gauss-Jacobi family Opsomer and Huybrechs [2023] to induce our family of OP (3) which have the form:

$$w(x) = (1-x)^\alpha (1+x)^\beta h(x); \quad x \in [-1, 1] \tag{4}$$

where $\alpha, \beta > -1$. $h(x)$ is the modifier over the standard Gauss-Jacobi weight function: $w(x) = (1-x)^\alpha (1+x)^\beta$. The only restriction on $h(x)$ is that it should be a strictly positive analytic function.

Cauchy Residue Theorem. The Cauchy Residue Theorem [Stein and Shakarchi, 2010] is a powerful tool to compute line integrals of analytic functions over closed curves. Let f be a function that is holomorphic on a simply connected open subset of the complex plane, except possibly at a finite set of points $a_1, \cdots, a_n$(called poles) and γ be a positively oriented simple closed curve, then we have:

$$\oint_\gamma f(z) \mathrm{d}z = 2\pi i \sum_{k=1}^{n} Res(f, a_k); \quad Res(f, a_k) = \frac{1}{2\pi i} \oint_{\gamma_k} f(z) \mathrm{d}z \tag{5}$$

where the quantity $Res(f, a_k)$ is the *complex residue* of the pole a_k. γ_k is a positively oriented simple closed curve around the pole a_k *not* including other singularities. This is the general formula; in our case, we will only deal with *simple poles*, so it simplifies [Stein and Shakarchi, 2010],

$$Res(f, a_k) = \lim_{z \to a_k} (z - a_k) f(z) \tag{6}$$

3 Strong and Weak Forms

We use u to denote the solution for a given PDE. Since, u is learned (parameterized using neural networks), it is commonly called the *trial function*. In its most generic form, a PDE is:

$$\mathcal{L}u = f, \quad \text{in} \quad \Omega; \quad u = g, \quad \text{in} \quad \partial\Omega \tag{7}$$

where, $\mathcal{L}$ denotes the differential operator acting on u, f denotes the non-homogeneity.. Also, $g, g_1, \ldots$ denote functions corresponding to the initial and/or boundary conditions as needed and Ω gives the domain of definition of the PDE and $\partial\Omega$ denotes its boundary.

Solving PDEs. We examine the canonical form of a second order elliptic PDE, the Poisson's equation (in 2 dimensions) as a running example. In 2D-Poisson's equation, the operator $\mathcal{L}$ is the Laplace operator, ∇^2. For $u : \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^2$ is the domain of interest this is given by:

$$-\nabla^2 u(x, y) = f(x, y), (x, y) \in \Omega; \quad u(x, y) = 0, (x, y) \in \partial\Omega \tag{8}$$

where f is the *forcing* function. We consider Ω to be $[-1, 1] \times [-1, 1]$ and $\partial\Omega$ as the natural boundary.

Before getting into details of our proposal, we summarize two solution approaches: via the *strong* and *weak* form of the PDE respectively. This will help us see how certain choices (such as **quadrature rule** and **collocation points**) will be key to our parameterization and thereby, learning.

Strong Form. Solving the PDE in its strong form (7) or (8) is equivalent to asking that the equations in (8) are satisfied *exactly* at several points along the domain Ω and boundary $\partial\Omega$. This can be done by sampling a large number of points distributed uniformly (Monte Carlo). While this approach is reasonable for simple cases, in more complex settings where the solution is not smooth, uniform sampling may lead to reduced approximation quality Lu et al. [2021].

Weak Form. While the strong form enforces point-wise exactness, we may only want the property to hold in a "weighted" sense for the entire function. This yields the *weak form*, which involves integration with a *test function*. For the 2D-Poisson equation, using a test function $v(x, y)$ and integrating over the domain, we have

$$\int\int_\Omega -\nabla^2 u \quad v \quad \mathrm{d}x\mathrm{d}y = \int\int_\Omega f \quad v \quad \mathrm{d}x\mathrm{d}y \tag{9}$$

Solving the PDE in its weak form moves the previous choice of points to a choice of *test functions*. The idea is to use a family of test functions based on the problem at hand and different methods emerge from this choice. These methods are called **Galerkin** methods. If one decides that the test functions in the weak form are Dirac-delta functions, then the Galerkin method reduces to **collocation** and the *weak form* becomes the *strong form* involving differential equations, which then need to be satisfied at the collocation points. When we discuss the *strong form*, we will interchangeably use "the choice of collocation points" and "the choice of test functions".

4 How to learn Quadrature Rules?

The above discussion underscored the importance of choice of *test functions* in the *weak form* or the choice of *collocation points* for solving the PDE in its *strong form*. One may ask: *can the underlying physics inform these choices?* To do so, we leverage the rich theory of OP.

Learning the weight function: We consider weight functions to be continuous and positive functions defined in some interval $\mathcal{I}$. Each such weight function $w(x)$ induces a family of OP given by (3). By learning the weight function for a set of OP, we want to enable a learnable (or adaptive) quadrature. For this to be practical, we want to compute these efficiently. Our method exploits asymptotic expansions of quadrature nodes for efficiency which are most complete for modified Gauss-Jacobi type weight function. So, we consider the modified Gauss-Jacobi form (4) and parameterize the modifier $h(x)$ in (4) using a neural network with parameters θ. This keeps the construction simple but offers many nice benefits we will see soon. So, our learnable weight function has the form:

$$w_\theta(x) = (1-x)^\alpha (1+x)^\beta h_\theta(x); \quad x \in \mathcal{I} \tag{10}$$

In (10), α and β can also be parameterized/learnt but in our experiments, we find that only learning $h_\theta(x)$ suffices. We consider the interval of the weight function $\mathcal{I}$ to be the same as Ω, the domain of the PDE. Next, we will see how this learnable weight function cleanly ties to the choice of test functions for the weak form and the collocation points in the strong form.

Remark 4.1. The method can, in principle, be extended to higher dimensions by treating each dimension independently and stacking the sampled values. Realizing such extensions in practice, however, would necessitate more elaborate constructions such as tensor-product formulations or sparse grid techniques [Garcke et al., 2006] to appropriately handle increased complexity.

Relation to Solving PDEs in Weak Form: For the weak form, consider a weighted integral using our weight function w_θ as the *test function* v, on both sides of (9),

$$\int\int_\Omega -\nabla^2 u \, w_\theta \, \mathrm{d}x\mathrm{d}y = \int\int_\Omega f \, w_\theta \, \mathrm{d}x\mathrm{d}y \tag{11}$$

We must compute both sides of the *integral* efficiently, especially for weight functions that produce the largest error in the integral equality. Our choice of weight functions crucially helps this computation.

Use of Orthogonal Polynomial & Quadrature Rule: Consider a one-dimensional integral. It is well-known that a n-point Gaussian **quadrature rule** can be constructed to yield a very good approximation to the integral of a $2n-1$ degree polynomial, multiplied with the corresponding weight function. For example, if we use the standard (non-modified) Gauss-Jacobi weight function:

$$w(x) = (1-x)^\alpha (1+x)^\beta; \quad x \in [-1,1]; \quad \alpha, \beta > -1 \tag{12}$$

Then, the n-th order approximation to the integral is given by:

$$\int_{-1}^{1} f(x)(1-x)^\alpha (1+x)^\beta \mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i) \tag{13}$$

where x_i denotes the i-th node and w_i the corresponding weight of the n point Gauss-Jacobi quadrature. Here, $f(x)$ is a smooth function on $[-1,1]$. The nodes x_i are in fact the ***roots*** of the n-th degree Jacobi polynomial, which form a family of OP w.r.t. the weight function in (12). Our learnable weight function not only provides a data-dependent choice of *test functions* associated with the *weak form* but also induces a ***learnable quadrature rule*** to compute the integral for the *weak form*!

Relation to Solving PDEs in Strong Form: Solving a PDE in its strong form means enforcing the relationship at several *collocation points*. A popular method in this category is *orthogonal collocation*

[Young, 2019] where the collocation points used are ***roots*** of ***orthogonal polynomials***. Thus, with our choice of *learnable weight function*, w_θ (which induces a family of OP (3)) also provides a data-dependent method to sample collocation points to solve a PDE in its strong form.

Remark 4.2. Standard quadratures with fixed nodes and weights are easy to implement in PINNs, but their lack of adaptivity leads to suboptimal residual minimization during training. Being data independent they cannot be conveniently utilized for solving a *family* of PDEs via neural networks.

5 Learning Quadrature Rules Efficiently

Solving PDEs via *strong form*, means that we must identify collocation points to evaluate the PDE. For the *weak form*, solving amounts to numerically approximating the integral via a quadrature rule. While §4 described the relation between learnable weight function with weak and strong form solution methods, it is not obvious how this can be done efficiently. Let us lay out the task. In strong form (7),(8), we must find the ***roots*** of corresponding OP. These will serve as the collocation points. In weak form, the learnable weight function(s) itself can act as test function(s). But we must find the quadrature rule (***roots*** of OP and corresponding ***quadrature weights***) to evaluate (11) efficiently. Thus, while the weight function is relevant by itself, our main interest is in the roots of OP induced by the weight function. Next, we show how use of recent advances in ***asymptotic expansions*** *of OP and their roots suggest an efficient instantiation.*

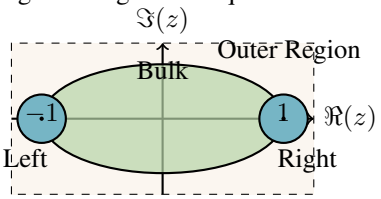


Figure 2: Different regions of complex plane for asymptotic expansion.

5.1 Instantiating Asymptotic Expansions

There is a mature literature for fast computation of quadrature nodes and weights corresponding to weight functions of OP [Townsend, 2015]. Over the last few years, this race is dominated by asymptotic expansions [Bogaert, 2014, Townsend et al., 2016]. Very recently, Opsomer and Huybrechs [2023] proposed *asymptotic expansions* for *generalized* (i.e., modified) versions of canonical weight functions including Gauss-Jacobi and Gauss-Hermite type.

Division of Complex Plane. We briefly present asymptotic expansions of nodes (roots of OP) and weights of quadrature rule for the modified Jacobi-type weight function (4) [Opsomer and Huybrechs, 2023, Opsomer, 2018]. The details are not crucial, but useful to appreciate our choice of parameterizations. The reader can check Appendix 9 and Opsomer and Huybrechs [2023] for more details on the expansions. The complex plane is divided into four regions (Fig.2) each with a different expansion. These are: the lens covering a bulk of the interval $(-1, 1)$, the two end-points referred to as left and right disks and everything else is the outer region. Let Γ be a shorthand notation for $(2n + \alpha + \beta + 1)$ and n refers to the polynomial degree.

Left endpoint. Truncated asymptotic expansions of nodes (x_k) and weights (w_k) near $x = -1$ are:

$$x_k \sim -1 + \frac{2j_{\beta,k}^2}{(\Gamma+d_0)^2} + \frac{-2j_{\beta,k}^2(j_{\beta,k}^2 - 3\alpha^2 - \beta^2 + 1)}{3(\Gamma + d_0)^4} + \dots; \quad \frac{w_k}{w(x_k)} \sim \frac{8}{J_{\beta-1}^2(j_{\beta,k})[\Gamma - d_0]^2} + \dots \tag{14}$$

where $w(x_k)$ is the value of the weight function at node x_k, and c_0 and d_0 are expansion coefficients, described shortly. The expansion uses both (a) the zeros of Bessel functions of order β denoted as $j_{\beta,k}$ (k-th zero) and the Bessel function of order $(\beta - 1)$ denoted by $J_{\beta-1}$.

Right endpoint. For the right end-point, we interchange α and β and use $h(-x)$ instead of $h(x)$

Bulk region. For expansions in the bulk region, we need to find the leading order term, t_k by solving:

$$\pi \frac{4k + 2\alpha + 3}{4k + 2\alpha + 2\beta + 2} = \arccos(t_k) + \frac{\sqrt{1 - t_k^2}}{\Gamma} \frac{1}{2\pi i} \oint_\gamma \frac{\log(h(\xi))\mathrm{d}\xi}{\sqrt{\xi^2 - 1}(\xi - t_k)} \tag{15}$$

Using t_k, the truncated asymptotic expansions of nodes and relative weights in the bulk region are:

$$\begin{aligned} x_k &\sim t_k + \frac{2\alpha^2 - 2\beta^2 + (2\alpha^2 + 2\beta^2 - 1)t_k}{2[\Gamma + \tau_0]^2} + \\ \frac{w_k}{w(x_k)} &\sim \frac{\pi\sqrt{1 - t_k^2}}{\Gamma}\Big[2 - \frac{2\tau_1(1 - t_k^2) - 2\tau_0 t_k}{\Gamma}\Big] + \dots \end{aligned} \tag{16}$$

where τ_0, τ_1 are also expansion coefficients.

Summary. In (14)–(16), the values of α, β correspond to the one used in the modified Gauss-Jacobi weight function (4). The value of n determines the degree of the OP $p_n(x)$ from the family (3) whose roots we want to compute. Finally, given n, $k \in \{1, 2, \ldots, n\}$ corresponds to the k-th root of polynomial $p_n(x)$, which is guaranteed to exist and be unique in the interval of definition.

Expansion Coefficients. In our description above, we used several coefficients: c_0, d_0 and τ_0, and τ_1. While more details are in Appendix 9, a synopsis is that the coefficients c_k, d_k stem from series expansion of the modulation function $h(x)$ in (4) (or the parameterized version in (10)) around $z = \pm 1$. The coefficients τ_i are the series coefficients resulting from the expansion of the contour integral in (15) around the leading order t_k of the k-th root of the orthogonal polynomial. The above formulas involve computation of contour integrals, root finding, and series expansions. Computing all these terms exactly within a learnable module will be challenging. We will next perform some simplifications so that the model is amenable to learning.

5.2 Simplifications, Assumptions and Implementation

We outline key assumptions, simplifications and implementation details for an efficient instantiation.

Simple Poles. Finding the leading order of the k-th root, t_k, requires solving a contour integral (15). We use the fact that roots in the bulk region are real and lie in $(-1, 1)$, and assume the integrand has only simple poles around t_k, allowing us to compute residues via (6) as follows:

$$\lim_{z \to t_k} (\xi - t_k) \frac{\log(h(\xi))}{\sqrt{1-\xi^2}(\xi - t_k)} = \frac{\log(h(t_k))}{\sqrt{1-t_k^2}} \tag{17}$$

Assuming real roots and simple poles, the Cauchy Residue Theorem (5), simplifies (15) as

$$\pi \frac{4k + 2\alpha + 3}{4k + 2\alpha + 2\beta + 2} = \arccos(t_k) - \frac{\log(h(t_k))}{\Gamma} \tag{18}$$

Root finding and Implicit Function Theorem. While we avoided computing the contour integral, we must solve (18) for t_k. Since t_k corresponds to the leading order of the *root* of an OP, it must exist in $(-1, 1)$. Thus, we use ***bisection method*** to find t_k via root finding of:

$$F(t_k) = \pi \frac{4k + 2\alpha + 3}{4k + 2\alpha + 2\beta + 2} - \arccos(t_k) + \frac{\log(h(t_k))}{\Gamma} \tag{19}$$

We use automatic implicit differentiation from [Blondel et al., 2022], which uses auto-diff of $F(t_k)$ and the implicit function theorem to automatically differentiate through the bisection method.

Benefits of simplifications & parameterization. We use fully connected layers followed by $tanh$ non-linearities to parameterize the solution function u, the modulating function h as well as to predict expansion coefficients of the nodes and weights, $c_0, d_0, d_1, \tau_0, \tau_1$, etc. for bulk and edge regions. The simplifications above offer multiple benefits. First, we avoid computing contour integrals within a differentiable learning framework. Second, we are able to compute all nodes and weights in **parallel** thereby making the process very *efficient* even for a very large number (*millions*) of nodes. It is worth noting that the exact procedure to compute the nodes (beyond the leading order term) in Section 4.2 of [Opsomer and Huybrechs, 2023] has a linear time complexity due to the several re-substitutions involved to find the coefficients. Empirically, we verify that the distribution of nodes and weights from the simplifications and parameterization choices does not hurt the distribution of quadrature which converges to the expected distribution.

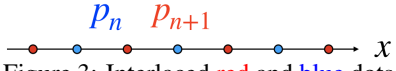


Figure 3: Interlaced red and blue dots on x axis correspond to the roots of polynomial $p_{n+1}(x)$ (degree $n+1$) and $p_n(x)$ (degree n) respectively.

Interlacing of roots of orthogonal polynomial. A naive application of quadrature nodes and weights for the OP is insufficient in several cases. This is because with a high degree polynomial (and so, a large number of nodes), within a few epochs, jointly training the solution and quadrature functions leads to over-fitting. Interestingly, this can be solved via a very useful property of the family of OP, namely ***interlacing*** of roots. Given the roots of an OP of degree $n+1$, the roots of the OP of degree n that belong to the same family are interlaced within the roots of p_{n+1} as shown in Fig. 3.

Thus, by utilizing quadrature nodes and weights stemming from varying degree of OP (all of whom correspond to the same weight function being learned), we introduce the desired stochasticity to prevent over-fitting.

Remark 5.1. In [Mishra and Molinaro, 2023], Theorem 2.6 decomposes the generalization error into training error E_T and a quadrature term $C_{pde}C_{quad}^{1/p}N^{-\alpha/p}$, where C_{quad} depends on the sampling distribution. LearnQuad directly targets C_{quad} by learning a sampling distribution that concentrates collocation points where the residual is high, without modifying the PINN architecture or PDE model.

Implementation Details: To ensure that our learned weight function is positive, we use *softplus* activation on the last layer of the network for $h_\theta(x)$. Further, the presence of $\log$ in (19) can lead to vanishing gradients, which we fixed by adding a small amount of noise (order of e^{-6}). We use the standard loss function used in PINN literature [Karniadakis et al., 2021, Cai et al., 2021] Additionally to avoid invalid quadrature rules from numerical issues, we use:

$$l_w = (\sum_{i=1}^{n} w_i - \int_{-1}^{1} w_\theta(x)\mathrm{d}x)^2 + (\sum_{i=1}^{n} w_i - 2)^2 \tag{20}$$

where the first term in (20) enforces the necessary condition that the sum of quadrature weights is equal to the integral of the weight function over the domain of definition. The second term in (20) discourages the quadrature weights from becoming too small. We note that the quadrature weights w_i are computed numerically from the learned weight function using asymptotic expansions and not directly learned.

Instead of integrating a PDE solver into the learning framework, the PINN loss is chosen due to challenges in differentiating through explicit/implicit solvers, including high memory usage, numerical instability, and computational overhead. Using PINN loss enables end-to-end learning and facilitates the development of learnable quadrature rules that generalize across different PDE instances. Our overall procedure has two trainable components: one is the learnable quadrature module (**LearnQuad**) and the other is the learnable solution function for the given PDE. These can be trained jointly using the loss described above either to simultaneously decrease it or in a min-max fashion where the quadrature module tries to provide hard-to-approximate function points in the domain. Empirically, we do not find a large difference in this specific choice of optimization. We provide pseudo-code for training PINN using LearnQuad in Algorithm 1. As demonstrated in Appendix 10.1.8 variations of α and β in the modified Gauss-Jacobi weight function only lead to minor variations in performance. So, we fix α, β to moderate values (typically in the range 1–2) for all experiments. This choice is sufficient to achieve the desired performance, without requiring extensive hyper-parameter sweeps. Our code is available at https://github.com/vsingh-group/learn-quad.

Algorithm 1 Training for a single PDE

1: **Input:** PDE parameter μ; #epoch: T, Learnable model u_ϕ, w_θ; Loss L incorporates operator $\mathcal{L}$, inhomogeneous term, initial/boundary condition; regularizer l_w.
2: **for** $i = 1$ **to** $i = T$ **do**
3: Use §5 to get quadrature nodes $\{x_l\}$
4: Use solution function u_ϕ on $\{x_l\}$
5: Loss:$l = L(u_\phi(x_l)) + l_w(w_\theta)$
6: Update u_ϕ and w_θ based on l
7: **end for**
8: **Output:** Learned models θ and ϕ

6 Experimental Evaluations

We demonstrate the effectiveness of our proposed framework involving the learnable quadrature module (**LearnQuad**) in solving PDEs via PINNs next. First we compare the empirical performance in solving single PDEs which demonstrate their effectiveness when used with PINNs. We emphasize that the use of **LearnQuad** improves the performance of PINNs and helps close the gap with numerical methods based solvers. Thereafter, we describe how the use of hyper-networks can enable

Table 1: **LearnQuad** has lowest L_2 relative error (mean $\pm$ standard deviation) in all PDEs.

	PDE # points	Burgers' 2000	Allen-Cahn 1000	Wave 2000
Non-adaptive	Grid	0.12 ± 0.04	0.88 ± 0.06	0.42 ± 0.09
	Random	0.13 ± 0.03	0.32 ± 0.14	0.48 ± 0.07
	LHS	0.18 ± 0.15	0.32 ± 0.04	0.61 ± 0.13
	Halton	0.06 ± 0.02	0.18 ± 0.05	0.46 ± 0.06
	Hammersley	0.07 ± 0.05	0.17 ± 0.05	0.31 ± 0.09
	Sobol	0.08 ± 0.03	0.20 ± 0.10	0.49 ± 0.09
Adaptive	Random-R	1.69 ± 1.67	0.55 ± 0.34	0.72 ± 0.90
	RAR-G	0.12 ± 0.04	0.53 ± 0.19	0.81 ± 0.11
	RAD	0.02 ± 0.00	0.08 ± 0.06	0.09 ± 0.04
	RAR-D	0.03 ± 0.01	0.09 ± 0.03	0.29 ± 0.04
	LearnQuad	$\mathbf{0.003 \pm 0.002}$	$\mathbf{0.03 \pm 0.008}$	$\mathbf{0.005 \pm 0.0006}$

Table 2: L_2 error with 1000 collocation points. **LearnQuad** yields best results in 4 of 5 cases.

PDE	**Convection** ($\beta = 30$)		**Convection** ($\beta = 50$)		**Allen Cahn**
Epochs.	100k	300k	150k	300k	200k
PINN (fixed)	$107.5 \pm 10.9\%$	$107.5 \pm 10.7\%$	$108.5 \pm 6.38\%$	$108.7 \pm 6.59\%$	$69.4 \pm 4.02\%$
PINN (dynamic)	$2.81 \pm 1.45\%$	$1.35 \pm 0.59\%$	$24.2 \pm 23.2\%$	$56.9 \pm 9.08\%$	$0.77 \pm 0.06\%$
Curr Reg	$63.2 \pm 9.89\%$	$2.65 \pm 1.44\%$	$48.9 \pm 7.44\%$	$31.5 \pm 16.6\%$	–
CPINN (fixed)	$138.8 \pm 11.0\%$	$138.8 \pm 11.0\%$	$106.5 \pm 10.5\%$	$106.5 \pm 10.5\%$	$48.7 \pm 19.6\%$
CPINN (dynamic)	$52.2 \pm 43.6\%$	$23.8 \pm 45.1\%$	$79.0 \pm 5.11\%$	$73.2 \pm 3.6\%$	$1.5 \pm 0.75\%$
RAR-G	$10.5 \pm 5.67\%$	$2.66 \pm 1.41\%$	$65.7 \pm 1.77\%$	$43.1 \pm 28.9\%$	$25.1 \pm 23.2\%$
RAD	$3.35 \pm 2.02\%$	$1.85 \pm 1.90\%$	$66.0 \pm 1.55\%$	$64.1 \pm 11.9\%$	$0.78 \pm 0.05\%$
RAR-D	$67.1 \pm 4.28\%$	$32.0 \pm 25.8\%$	$82.9 \pm 5.96\%$	$75.3 \pm 9.58\%$	$51.6 \pm 0.41\%$
L^∞	$66.6 \pm 2.35\%$	$41.2 \pm 27.9\%$	$76.6 \pm 1.04\%$	$75.8 \pm 1.01\%$	$1.65 \pm 1.36\%$
R3	$1.51 \pm 0.26\%$	$0.78 \pm 0.18\%$	$1.98 \pm 0.72\%$	$2.28 \pm 0.76\%$	$\mathbf{0.83 \pm 0.15\%}$
Causal R3	$2.12 \pm 0.67\%$	$0.75 \pm 0.12\%$	$5.99 \pm 5.25\%$	$2.28 \pm 0.76\%$	$\mathbf{0.71 \pm 0.007\%}$
LearnQuad	$\mathbf{0.78 \pm 0.002\%}$	$\mathbf{0.68 \pm 0.02\%}$	$\mathbf{0.79 \pm 0.02\%}$	$\mathbf{0.76 \pm 0.01\%}$	$0.87 \pm 0.01\%$

LearnQuad to efficiently solve multiple instances of a given PDE. Our extensive set of baselines include: PINN(fixed), PINN(dynamic) [Karniadakis et al., 2021], Curr Reg [Krishnapriyan et al., 2021], CPINN (fixed, dynamic) Wang et al. [2022], RAR-G [Lu et al., 2021], RAD [Nabian et al., 2021], RAR-D [Wu et al., 2023], R3 and Causal R3[Daw et al., 2023]. We performed experiments on benchmark PDEs considered in the baselines. This ensured that we are able to clearly demonstrate the advantage of **LearnQuad**.

6.1 Solving PDEs using LearnQuad

Setup: We compare the performance of **LearnQuad** in PINN-based solutions of several well known PDEs. We benchmark the performance of our proposed data adaptive quadrature scheme against several other adaptive and non-adaptive algorithms [Wu et al., 2023, Lu et al., 2021, Daw et al., 2023]. Additional experimental details including the explicit form of the PDEs, hyper-parameter details used in the experiment are included in Appendix 10.1. We used the exact same number of points to train the solution model in all methods for a given PDE, and the number of collocation points is chosen based on the baseline methods. The solution model in each case had the exact same number of parameters to ensure a fair comparison.

Result: We report L_2 relative error (48) as the performance metric following two different experimental settings from Daw et al. [2023] and Wu et al. [2023] over 7 PDEs in Table 2 and Table 1 respectively. In all but one scenario, **LearnQuad** is able to achieve the best solution function. As can be seen from the numerical results in Table 2 and 1, the L_2 relative error for models trained using LearnQuad are better by an order of magnitude in most cases and also have very little variance (results reported are an average over five runs). The performance of LearnQuad improves as the number of evaluation points increases, as presented in Table 6. Additionally, we observe that LearnQuad can achieve similar performance to other adaptive methods with a much smaller number of points in many cases. All methods have a comparable runtime and memory consumption.

Summary: As an adaptive method, **LearnQuad** is highly effective in PINN-style solving of PDEs, leading to performance boost of PINNs in all cases. The findings highlight the usefulness of adaptive methods over non-adaptive ones specifically when the solution function is not well-behaved. **LearnQuad** can be used as drop in replacement for any sampling strategy.

Remark 6.1. We include additional results on solving PDEs via LearnQuad in their strong form, weak form and also using the energy method in Appendix 10.3. These demonstrate that **LearnQuad** is a versatile adaptive scheme which can be used to solve PDEs in multiple reformulations.

Remark 6.2. We use *LearnQuad* in solving a 100 dimensional PDE, a Poisson equation with a very smooth solution (details in Appendix 10.2 following Yu et al. [2018]) and achieve a relative L_2 error of 0.085 which is similar to using naive Monte Carlo in this setting with relative L_2 error of 0.09. This illustrates the viability of *LearnQuad* for high dimensional PDEs. Since the solution is smooth in this particular case, there is no substantial benefit in using a data-driven adaptive method.

Remark 6.3. In Appendix 10.1.7 we show that LearnQuad remains numerically stable and improves performance by increasing the number of collocation points. We observe efficient scaling for number of collocation points — runtime remains nearly flat up to large point sets, with only modest sub-

linear growth at very large scales due to hardware bandwidth limits (Appendix 11.1). We also discuss runtime comparison of introducing LearnQuad in PINNs in Appendix 11.2. While our Jax implementation is the fastest among adaptive alternatives, compared to vanilla PINNs LearnQuad is slightly more expensive due to an additional compute operations.

6.2 Comparison of LearnQuad to Classical Solvers

Setup: We benchmark strong non-PINN solvers on multiple problems and report the results alongside adaptive PINN solvers, including **LearnQuad**. We used the PDEs from the Section 6.1, the details of which are in Appendix 10.1. The relative L_2 error is computed against the true analytical solution which is readily available in these cases.

Result: As expected, standard methods (e.g., Lax-Wendroff, Crank-Nicolson), achieve the lowest relative errors as shown in Table 3. However, LearnQuad consistently improves PINN performance across all cases and notably outperforms the strongest adaptive PINN baselines.

Summary: LearnQuad advances PINN performance over existing sampling methods, though there still exists a gap with classical solvers. We also acknowledge that PINNs in their current state are typically more expensive and less accurate than classical solvers for forward problems. The results do not suggest superiority over numerical methods but helps put the scale of gains **LearnQuad** brings for PINNs in context.

Table 3: L_2 Relative Error Comparison: Numerical Solvers vs. Adaptive PINNs. Best performing numerical methods are in purple and most competitive adaptive PINN methods are in blue. LearnQuad (LQ) improves performance over other adaptive methods, but there remains a performance gap with numerical solvers. Legends: LW = Lax-Wendroff; BW = Beam-Warming; FE = Forward Euler; BE = Backward Euler; CN = Crank-Nicolson; LF = Leapfrog; CFD = Centered FDM; NB = Newmark-beta; R3 [Daw et al., 2023], RAR-G [Lu et al., 2021], RAD[Nabian et al., 2021].

PDE	**Numerical Solvers**			**Adaptive PINNs**	
	Method / L_2 Error			**Method / L_2 Error**	
Convection ($\beta = 30$)	Upwind 0.0460	LW 0.0002	BW 0.0002	R3 0.0078	LQ 0.0068
Convection ($\beta = 50$)	Upwind 0.0755	LW 0.0003	BW 0.0003	R3 0.0228	LQ 0.0076
Diffusion	FE 0.0004	BE 0.0004	CN 0.0004	RAR-G 0.0009	LQ 0.0005
Wave	LF 0.0021	CFD 0.0021	NB 0.0015	RAD 0.0900	LQ 0.0050

6.3 Solving a family of PDEs via LearnQuad

Setup: Given the effectiveness of **LearnQuad** in solving a given PDE, we now utilize our framework to tackle a harder problem. We consider a *family* of PDEs, where our end goal is to solve a PDE given a particular choice of forcing function and/or PDE hyper-parameters and initial and/or boundary conditions. Based on (7), a family of PDEs corresponding to differential operator $\mathcal{L}$ refers to the set of triplets, $\{(f_i, g_i, u_i)\}_{i=1}^{N}$, where each i-th PDE satisfies $\mathcal{L}u_i = f_i, \quad \text{in} \quad \Omega; u_i = g_i, \text{in} \quad \partial\Omega$. We only assume access to f_i, g_i's and $\mathcal{L}$. We note that f_i denotes the forcing function and/or PDE hyper-parameter and g_i denotes the initial and/or boundary condition corresponding to the i-th PDE which is governed by operator $\mathcal{L}$. As an example, if f has the following parametric form:

$$f_\kappa(x) = -(a(\pi\theta)^2 \sin(\pi\theta x) + b(\pi\psi)^2 \cos(\pi\psi x)); \quad \text{where} \quad \kappa = \{a, b, \theta, \phi\} \sim p \tag{21}$$

we sample $\kappa \sim p$ to obtain f_i's (similarly for g_i's) and learn to solve for PDEs corresponding to $\mathcal{L}$. We use two hyper-networks with parameter(s) θ and ϕ which provide parameters of the weight function $w_\kappa(x)$ and solution function $u_\kappa(x)$ respectively based on the input $\kappa \sim p$. This weight function is then used to generate a suitable quadrature $\{x_l\}_\kappa$ for the PDE corresponding to κ. These are then used to evaluate the $\mathcal{L}u_\kappa(x)$ and $f_\kappa(x)$ and minimize the loss based on the strong form as in Fig. 4. Pseudo-code and experimental details are in Appendix 10.4. We demonstrate the effectiveness of *LearnQuad* via: Laplace, Advection, Burger's, Wave and Heat equation.

Table 4: Absolute relative error on the test set of four different PDEs. Details are in Appendix 10.4.

PDE	Wave		Advection		Heat		Burgers'	
IC/BC	(70)	(71)	(74)	(75)	(65)	(66)	(80)	(81)
Error	9.9e-6	3.3e-5	1.9e-5	7.9e-5	2.1e-4	3.5e-4	2.8e-4	3.3e-4

Result: We report the absolute relative error compared to the numerical solution obtained using the same number of domain points used for *LearnQuad* in Table 4. As can be seen, the model generalizes very well to test set PDEs. Additional visualizations are included in Appendix 10.4.

Summary: Learning quadratures for a family of PDEs is beneficial. Once trained, given a new forcing function and boundary/initial condition can generate the solution in a single forward pass. This is *highly efficient* in terms of time and eliminates the need to store or process separate solutions, unlike standalone PINNs.

Remark 6.4 (Distinction with Operator Learning). While the end result of learning for a family of PDEs is similar to operator learning, the setting is different. While operator learning uses paired data (f_i, u_i), PINNs with **LearnQuad** *only* utilize f_i's and shared operator $\mathcal{L}$ without requiring any problem-solution paired data.

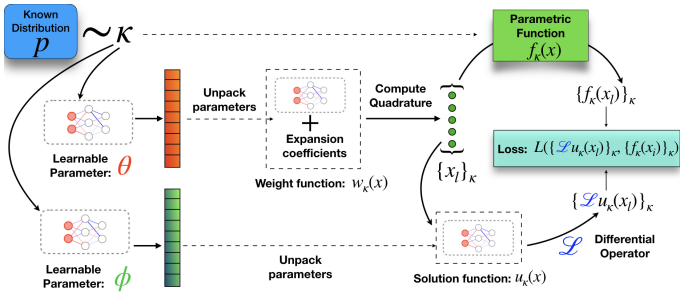


Figure 4: Solving a family of PDEs governed by operator $\mathcal{L}$, forcing function/external condition/PDE hyper-parameter are parameterized by known distribution p. Two hyper-networks with parameters θ and ϕ generate the weight and solution function.

7 Related Work

Beyond the literature described in §1, a large body of work focuses on discovering solutions to PDEs using neural networks. **LearnQuad** enables PINNs Raissi et al. [2019], to use data-dependent sampling of collocation points. While Variational-PINN [Kharazmi et al., 2019] and hp-VPINN [Kharazmi et al., 2021] solve PDEs in weak form, they use a careful choice of test functions, which is learnable in our case. Compared to Deep-Ritz [Yu et al., 2018], our method does not need the minimum energy principle to be applicable. Our work provides a novel way to improve different variants of PINNs and is complementary to existing literature. We acknowledge recent ideas adjacent to adaptive quadrature [Rivera et al., 2022, Omella and Pardo, 2024, Lau et al., 2024], which either directly try to optimize node locations thereby resulting in a much larger optimization problem or fall-back to problem-specific regularizer(s) which limit their applicability. The learnable weight function in **LearnQuad** induces problem-specific quadrature rules. It differs from classical adaptive methods and ML-based techniques like R3 [Daw et al., 2023], RAR [Lu et al., 2021], RAD [Wu et al., 2023] which rely on residual-based error estimators or gradient thresholding. Other works like [Wang et al., 2021] introduce an annealed learning rate and a refined network architecture; [Urbán et al., 2025] proposed a modified optimizer and an adjusted loss. LearnQuad is orthogonal to these techniques—focusing on adaptive sampling and hence can be seamlessly integrated with them.

8 Conclusions

We present a data-driven approach to solve PDEs via PINNs, by exploiting new results of fast quadrature computation using asymptotic expansions and recent capabilities of implicit function differentiation. We demonstrate that incorporating our learnable quadrature scheme, **LearnQuad** within PINNs can lead to performance improvement over exisiting adaptive and non-adaptive sampling schemes across a diverse set of PDEs. Additionally, we show that incorporation of **LearnQuad** is quite beneficial when solving a family of PDEs – where the alternative would be to deploy a Monte Carlo based scheme for each instance individually. Our hyper-network based approach generates the solution to a PDE instance from a given family in just a single forward pass without requiring any additional training.

Limitations and Broader Impact: LearnQuad improves the capabilities of PINNs on commonly studied benchmark PDEs described in the literature. However, we should acknowledge that despite active work surrounding PINNs, in their current stage, these models are not a drop-in substitute for classical solvers for common real-world PDEs. This gap will likely get smaller with time but this context should clarify that PINNs (with or without LearnQuad) are not intended to substitute the mature body of existing work on traditional numerical methods algorithms and efficient implementations deployed across many fields. Extending LearnQuad to effective higher dimensional settings will likely involve the use of tensor products or sparse grids and is a ripe direction for future work. Our work describes mathematical development to improve PINNs, and as such, does not have an immediate societal impact.

Acknowledgments: Our work was motivated by interesting discussions with Daryl Nazareth (Roswell Park Cancer Institute) around the use of physics informed models [Zhou et al., 2023] for radiation therapy [Bedford, 2019]. We are grateful to Daryl and Anant Gopal (also at RPCI) for their time. We thank Peter Opsomer and Daan Huybrechs for their helpful discussion during the initial phase of this project and Lopamudra Mukherjee for feedback on a first draft of this paper. S.P. and V.S. were partly supported by a contract via the DARPA Strengthen program.

References

Mohammad H Aliabadi and David P Rooke. *Numerical fracture mechanics*, volume 8. Springer Science & Business Media, 1991.

William F Ames. *Numerical methods for partial differential equations*. Academic press, 2014.

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, pages 1–9, 2024.

WA Beckham, PJ Keall, and JV Siebers. A fluence-convolution method to calculate radiation therapy dose distributions that incorporate random set-up error. *Physics in Medicine & Biology*, 47(19): 3465, 2002.

James L Bedford. Calculation of absorbed dose in radiotherapy by solution of the linear boltzmann transport equations. *Physics in Medicine & Biology*, 64(2):02TR01, 2019.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *Advances in neural information processing systems*, 35:5230–5242, 2022.

Ignace Bogaert. Iteration-free computation of gauss–legendre quadrature nodes and weights. *SIAM Journal on Scientific Computing*, 36(3):A1008–A1026, 2014.

Nicolas Boullé and Alex Townsend. A mathematical guide to operator learning. *arXiv preprint arXiv:2312.14688*, 2023.

Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12): 1727–1738, 2021.

Stéphane d'Ascoli, Sören Becker, Alexander Mathis, Philippe Schwaller, and Niki Kilbertus. Odeformer: Symbolic regression of dynamical systems with transformers. *arXiv preprint arXiv:2310.05573*, 2023.

Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Mitigating propagation failures in physics-informed neural networks using retain-resample-release (R3) sampling. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Jochen Garcke et al. Sparse grid tutorial. *Mathematical Sciences Institute, Australian National University, Canberra Australia*, 7, 2006.

Gene H Golub and John H Welsch. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

Samuel Holt, Zhaozhi Qian, and Mihaela van der Schaar. Deep generative symbolic regression. *arXiv preprint arXiv:2401.00282*, 2023.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.

Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in neural information processing systems*, 34:26548–26560, 2021.

J Nathan Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017.

Gregory Kang Ruey Lau, Apivich Hemachandra, See-Kiong Ng, and Bryan Kian Hsiang Low. Pinnacle: Pinn adaptive collocation and experimental points selection. *arXiv preprint arXiv:2404.07662*, 2024.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020a.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2020b.

Qin Lou, Xuhui Meng, and George Em Karniadakis. Physics-informed neural networks for solving forward and inverse flow problems via the boltzmann-bgk formulation. *Journal of Computational Physics*, 447:110676, 2021.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021.

Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating pdes. *IMA Journal of Numerical Analysis*, 43(1):1–43, 2023.

Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, 36(8):962–977, 2021.

Sheehan Olver, Richard Mikaël Slevinsky, and Alex Townsend. Fast algorithms using orthogonal polynomials. *Acta Numerica*, 29:573–699, 2020.

Ángel J Omella and David Pardo. r-adaptive deep learning method for solving partial differential equations. *Computers & Mathematics with Applications*, 153:33–42, 2024.

Peter Opsomer. Asymptotics for orthogonal polynomials and high-frequency scattering problems. 2018.

Peter Opsomer and Daan Huybrechs. High-order asymptotic expansions of gaussian quadrature rules with classical and generalized weight functions. *Journal of Computational and Applied Mathematics*, 434:115317, 2023.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

JR Rice and Dennis Michael Tracey. Computational fracture mechanics. In *Numerical and computer methods in structural mechanics*, pages 585–623. Elsevier, 1973.

Jon A Rivera, Jamie M Taylor, Ángel J Omella, and David Pardo. On quadrature rules for solving partial differential equations using neural networks. *Computer Methods in Applied Mechanics and Engineering*, 393:114710, 2022.

Elias M Stein and Rami Shakarchi. *Complex analysis*, volume 2. Princeton University Press, 2010.

Alex Townsend. The race for high order gauss–legendre quadrature. *SIAM News*, 48:1–3, 2015.

Alex Townsend, Thomas Trogdon, and Sheehan Olver. Fast computation of gauss quadrature nodes and weights on the whole real line. *IMA Journal of Numerical Analysis*, 36(1):337–358, 2016.

Lloyd N Trefethen and David Bau. *Numerical linear algebra*, volume 181. Siam, 2022.

Jorge F Urbán, Petros Stefanou, and José A Pons. Unveiling the optimization process of physics informed neural networks: How accurate and competitive can pinns be? *Journal of Computational Physics*, 523:113656, 2025.

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

Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*, 2022.

Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, 2023.

Larry C Young. Orthogonal collocation revisited. *Computer Methods in Applied Mechanics and Engineering*, 345:1033–1076, 2019.

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

Jiahang Zhou, Ruiyang Li, and Tengfei Luo. Physics-informed neural networks for solving time-dependent mode-resolved phonon boltzmann transport equation. *npj Computational Materials*, 9 (1):212, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have presented experimental evidence to back our claims in Section 6. The abstract and introduction have well defined the scope of these experiments and set the tone.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations of our work in Section 8

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: We do not provide a theoretical result in the paper. Hence, this question is not applicable here.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Yes, we include all experimental details necessary to reproduce in Section 6 and Appendix 10. Pseudo code for the main algorithms presented are in Algorithm 1 and Algorithm 2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Yes, our code will be released when accepted for publication.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide extensive details for experiments in Section 6 and in Appendix 10.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our performance is reported as an average of several runs along with mean and standard deviation as is standard in relevant literature.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: These details are available in the Appendix 10.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, the research performed conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Yes, we include this in Section 8.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper doesn't invlove the release of pre-trained models or datasets.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, all sources have been cited appropriately.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release any new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The work presented does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendix

9 Asymptotic Expansion

In this section, we list the full expansion of nodes and weights used for experiments in the paper in Section 5:

For the left hard edge at $x = -1$ used in ((14))

$$
\begin{aligned}
&x_k \sim -1 + \frac{2j_{\beta,k}^2}{(\Gamma+d_0)^2} + \frac{-2j_{\beta,k}^2}{3(\Gamma+d_0)^4}[j_{\beta,k}^2 - 3\alpha^2 - \beta^2 + 1] + \dots \\
&+ \frac{-j_{\beta,k}^2}{6(2n+\alpha+\beta+1+d_0)^5}[16(d_0-3d_1)j_{\beta,k}^4 + 3(4\alpha^2-1)c_0 + (12\alpha^2+8\beta^2-5)d_0 \\
&- 6(4\beta^2-1)d_1] + \dots + O(n^{-8}) \\
&\frac{w_k}{w(x_k)} \sim \frac{8}{J_{\beta-1}^2(j_{\beta,k})[\Gamma-d_0]^2} \\
&+ \frac{8}{3J_{\beta-1}^2(j_{\beta,k})[2n+\alpha+\beta+1-d_0]^4}[3\alpha^2+\beta^2-1-2j_{\beta,k}^2] \\
&- \frac{2[32(d_0-3d_1)j_{\beta,k}^2 + 3(4\alpha^2-1)c_0 + (12\alpha^2+8\beta^2-5)d_0 - 6(4\beta^2-1)d_1]}{3J_{\beta-1}^2(j_{\beta,k})[2n+\alpha+\beta+1-d_0]^5} + \dots + O(n^{-8})
\end{aligned}
\tag{22}
$$

where $\Gamma = 2n+\alpha+\beta+1$

For the bulk region used in ((16)):

$$
\begin{aligned}
x_k \sim t_k + \frac{2\alpha^2-2\beta^2+(2\alpha^2+2\beta^2-1)t_k}{2[\Gamma+\tau_0]^2} - \frac{1}{4[2n+\alpha+\beta+1+\tau_0]^3}\big(4(\alpha^2-1)c_0 + \\
+4(\beta^2-1)d_0 + 8(\alpha^2-\beta^2)\tau_0 - 4(\alpha^2-\beta^2)\tau_1 + 2(2\alpha^2+2\beta^2-1)\tau_1 t_k^3 \\
+2[(2\alpha^2+2\beta^2-1)\tau_0 + 2(\alpha^2-\beta^2)\tau_1]t_k^2 \\
+[4(\alpha^2-1)c_0 - 4(\beta^2-1)d_0 + 4(3\alpha^2+\beta^2-1)\tau_0 - 2(2\alpha^2+2\beta^2-1)\tau_1]t_k\big) + h.o.t. \\
\frac{w_k}{w(x_k)} \sim \frac{\pi\sqrt{1-t_k^2}}{\Gamma}\big[2 - \frac{2\tau_1(1-t_k^2)-2\tau_0 t_k}{\Gamma}] + \frac{1}{(2n+\alpha+\beta+1)^2}\big(2\tau_1^2 t_k^4 + 4\tau_0\tau_1 t_k^3 \\
-4\tau_0\tau_1 t_k + 2(\tau_0^2-2\tau_1^2)t_k^2 + 2\alpha^2 + 2\beta^2 + 2\tau_1^2 - 1\big) + h.o.t.\big]
\end{aligned}
\tag{23}
$$

The coefficients c_k and d_k are given by:

$$
c_k = \frac{1}{2\pi i}\oint_\gamma \frac{\log(h(\xi))}{(\xi^2-1)^{1/2}}\frac{\mathrm{d}\xi}{(\xi-1)^{k+1}} \tag{24}
$$

$$
d_k = \frac{1}{2\pi i}\oint_\gamma \frac{\log(h(\xi))}{(\xi^2-1)^{1/2}}\frac{\mathrm{d}\xi}{(\xi+1)^{k+1}} \tag{25}
$$

The coefficients τ_0 and τ_1 are expansion coefficients in the following:

$$
\frac{1}{2\pi i}\oint_\gamma \frac{\log h(\zeta)d\zeta}{\sqrt{\zeta^2-1}(\zeta-z)} \sim \sum_{i=0}^{\infty}\tau_i(z-t_k)^i \tag{26}
$$

Hereafter, we refer the reader to [Opsomer, 2018, Opsomer and Huybrechs, 2023] for further detail on the asymptotic expansions pertinent to modified Gauss-Jacobi weight functions.

In Fig. 5 we demonstrate how different choices for $h_\theta(x)$ lead to different $w_\theta(x)$ and how these modifications are different from the standard Gauss-Jacobi weight function.

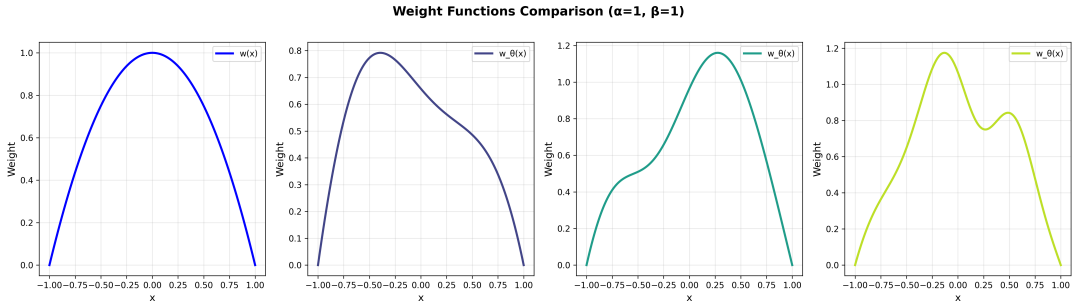


Figure 5: Comparison of weight functions. The left most plot is for the standard Gauss-Jacobi weigh function. The remaining plots are from different neural network parameterization of the modifier $h_\theta(x)$. One can see that the weight function changes significantly when optimizing for the modified Gauss-Jacobi function. In every case, we fixed α and β to be 1 and note that varying these is not necessary for our desired use-case.

10 Experiment Details

10.1 Solving PDEs via LearnQuad

Below we describe the PDEs used in the experimental results of Table 2 and Table 1.

10.1.1 Convection Equation

For the Convection equation in Table 2, we considered the following PDE:

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, \quad x \in [0, 2\pi],\ t \in [0, 1] \tag{27}$$

$$u(x, 0) = h(x) \tag{28}$$

$$u(0, t) = u(2\pi, t) \tag{29}$$

The model used in this case is a fully connected neural network with hidden layers of width 50 and depth 4.

10.1.2 Diffusion Equation

We consider the following one dimensional diffusion equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + e^{-t}\left(-\sin(\pi x) + \pi^2 \sin(\pi x)\right), \quad x \in [-1, 1],\ t \in [0, 1], \tag{30}$$

$$u(x, 0) = \sin(\pi x), \tag{31}$$

$$u(-1, t) = u(1, t) = 0, \tag{32}$$

with domain $[-1, 1]$ in space and $[0, 1]$ in time. The exact solution to this diffusion equation is given by $u(x, t) = \sin(\pi x)e^{-t}$, which is a smooth one and hence all methods as illustrated in Table 1 perform reasonably well even with a small number of points. The model used in this case is a fully connected neural network with hidden layers of width 32 and depth 3.

10.1.3 Burger's Equation

We consider the following Burger's equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1],\ t \in [0, 1], \tag{33}$$

$$u(x, 0) = -\sin(\pi x), \tag{34}$$

$$u(-1, t) = u(1, t) = 0, \tag{35}$$

where ν is the viscosity of the fluid and u is the desired flow velocity. In our experiments, we have used $\nu = 0.01/\pi$ which results in a non-smooth solution. The model used in this case is a fully connected neural network with hidden layers of width 64 and depth 3.

Table 5: L_2 relative error (mean $\pm$ standard deviation) of the trained solution function obtained while using different adaptive and non-adaptive methods. The lowest error for each problem is denoted in boldface. Model trained via **LearnQuad** achieves the lowest L_2 relative error in all case.

	PDE	Diffusion
	# points	30
Non-adaptive	Grid	0.004 ± 0.001
	Random	0.005 ± 0.002
	LHS	0.003 ± 0.002
	Halton	0.002 ± 0.0006
	Hammersley	0.001 ± 0.0007
	Sobol	0.002 ± 0.002
Adaptive	Random-R	0.12 ± 0.06
	RAR-G	0.0009 ± 0.0008
	RAD	0.0019 ± 0.00097
	RAR-D	0.004 ± 0.0041
	LearnQuad	$\mathbf{0.0005 \pm 0.0001}$

10.1.4 Allen-Cahn Equation

The Allen-Cahn PDE considered in our experiments and reported in Table 1 is as follows:

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + 5(u - u^3), \quad x \in [-1, 1],\ t \in [0, 1], \tag{36}$$

$$u(x, 0) = x^2 \cos(\pi x), \tag{37}$$

$$u(-1, t) = u(1, t) = -1, \tag{38}$$

We use a value of $D = 0.001$ as the diffusion coefficient in the PDE. The model used in this case is a fully connected neural network with hidden layers of width 64 and depth 3.
The Allen-Cahn PDE considered in Table 2 is as follows:

$$\frac{\partial u}{\partial t} - 0.0001\frac{\partial^2 u}{\partial x^2} + 5u^3 - 5u = 0, \quad x \in [-1, 1],\ t \in [0, 1] \tag{39}$$

$$u(x, 0) = x^2 \cos(\pi x) \tag{40}$$

$$u(t, -1) = u(t, 1) \tag{41}$$

$$\left.\frac{\partial u}{\partial t}\right|_{x=-1} = \left.\frac{\partial u}{\partial t}\right|_{x=1} \tag{42}$$

The model used in this case is a fully connected neural network with hidden layers of width 128 and depth 4.

10.1.5 Wave Equation

We consider the following one dimensional wave equation:

$$\frac{\partial u}{\partial t} = c^2\frac{\partial^2 u}{\partial x^2}, \quad x \in [0, 1],\ t \in [0, 1], \tag{43}$$

$$u(0, t) = u(1, t) = 0, \quad t \in [0, 1] \tag{44}$$

$$u(x, 0) = \sin(\pi x) + \frac{1}{2}\sin(4\pi x), \quad x \in [0, 1] \tag{45}$$

$$\frac{\partial u}{\partial t}(x, 0) = 0, \quad x \in [0, 1] \tag{46}$$

with $c = 2$, where c is the velocity of the wave. The solution in this specific choice demonstrates a multi-scale behavior in both space and time dimension and is as follows:

$$u(x, t) = \sin(\pi x)\cos(2\pi t) + \frac{1}{2}\sin(4\pi x)\cos(8\pi t) \tag{47}$$

The model used in this case is a fully connected neural network with hidden layers of width 100 and depth 5.

10.1.6 Other Details

The number of parameters used for the learnable weight function in the LearnQuad module was roughly 500 parameters in all cases. All neural networks were implemented using fully connected layers with $\tanh$ as the activation function. All experiments were performed on a single NVIDIA 2080 Ti GPU. The number of epochs used for diffusion PDE was $100k$ while for Burger's, Wave and Allen-Cahn PDE they were run for $200k$ epochs. This was determined empirically based on convergence of the L_2 relative error. We used a learning rate of 1e-3. As noted in Algorithm 1, one could either use a noise sampled from the standard normal or the PDE specific parameters as an input to the learnable quadrature module and results are not too different, but slightly better on using standard normal noise as input. We find jointly optimizing both the LearnQuad and solution model provides very good performance without the need for a sophisticated min-max optimization scheme. The L_2 relative error reported in the paper is computed as the following:

$$L_{2error} = \frac{||u_\theta - u||_2}{||u||_2} \tag{48}$$

Here, u_θ is the learned solution function and u is the "ground truth" solution. In a small number of cases where the true solution is available in a closed form we use that as u or we use u to be a numerical solution achieved using a traditional numerical scheme (finite difference). In any case, the test error is evaluated on a uniform grid of a much higher density $(10x)$ than the number of points used in the training scenario. We emphasize that the "ground truth" solution is not used in any form during the training period.

10.1.7 Performance of LearnQuad

We enumerate the performance of **LearnQuad** with increasing number of points in three different PDEs, (outlined previously) in Table 6. As expected, the performance in terms of L_2 relative error improves on increasing the number of points. Note that the solution to the diffusion equation is very smooth and hence even a very small number of points can lead to very good performance.

Table 6: L^2 Relative Error for Different PDEs with varying number of points used by LearnQuad. Performance improves on increasing the number of points as expected.

Diffusion Equation		Allen-Cahn Equation		Wave Equation	
No. of Points	L^2 Error	No. of Points	L^2 Error	No. of Points	L^2 Error
20	0.0013			200	0.017
25	0.0007	200	0.0444	500	0.0076
30	0.0004	700	0.0331	1500	0.0064
35	0.0003			2500	0.0052
40	0.0002	1500	0.0280	3500	0.0044

10.1.8 Performance of LearnQuad with varying hyper-parameter

We investigate the performance of LearnQuad with varying the hyper-parameters of α and β in the modified Gauss-Jacobi weight function from ((10)). We report the test performance in terms of the relative L^2 relative error in Table 7. We observe minor variations in the performance of LearnQuad based on the choice of these hyper-parameters.

Table 7: L_2 Relative Error for Different PDEs with varying α and β in the modified Gauss-Jacobi weight function used by LearnQuad. We observe there are minor variations based on the choice of these hyper-parameters.

(α, β)	**Diffusion**	**Wave**	**Convection**
$(2, 2)$	0.0004	0.0058	0.7299
$(3, 3)$	0.0005	0.0056	0.7163
$(1, 2)$	0.0004	0.0065	0.7207
$(2, 1)$	0.0006	0.0044	0.7323
$(10, 10)$	0.0007	0.0062	0.6774

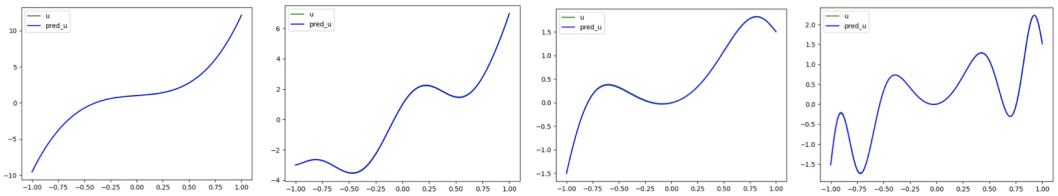


Figure 6: True/Predicted solution functions. 2 right-most two plots for 2 different conditions on the 1D-Laplace operator. 2 left-most two plots for solutions to 2 settings for the operator $\frac{\mathrm{d}^2}{\mathrm{d}x^2} + \frac{\mathrm{d}}{\mathrm{d}x}$

10.2 Solving high dimensional PDE

We consider the following high-dimensional Poisson equation.

$$-\Delta u = -200, \quad x \in (0,1)^{100} \tag{49}$$

$$u(x) = \sum_{i=1}^{100} x_i^2, \quad x \in \partial(0,1)^{100} \tag{50}$$

which is in a 100 dimensional space with the true solution being $u(x) = \sum_{i=1}^{100} x_i^2$. This is a Poisson equation with a very smooth solution [Yu et al., 2018]. LearnQuad achieves a relative L_2 error of 0.085 which is similar to using naive Monte Carlo in this setting with relative L_2 error of 0.09. This illustrates the viability of LearnQuad for high dimensional PDEs. Since the solution is smooth in this particular case, there is no substantial benefit in using a data-driven adaptive method. This experiment, demonstrates that LearnQuad is not restricted to low dimensional problems. We used a fully connected neural network with hidden layers having a depth of 3 and width of 100 as the solution model with $\tanh$ as the activation function. We used 1000 points in 100 dimensions. For this problem, our training took 18 seconds to converge in 300 epochs. After this, evaluating the trained model on any given resolution takes 0.0065 seconds. The test errors were computed with respect to the true analytical solution which is readily available in this case.

10.3 Solving PDEs in strong, weak and energy from via LearnQuad

We describe empirical evaluations of our proposed framework using **LearnQuad**. We show results for solving a single given PDE via all three main approaches: (a) the strong form, (b) weak form and (c) minimum principle.

(A) Numerical experiments with Strong Form. We begin by deploying our learnable quadrature first in solving PDEs via the strong from described in §3. We consider two operators: (a) 1D-Laplace and (b) $\frac{\mathrm{d}^2}{\mathrm{d}x^2} + \frac{\mathrm{d}}{\mathrm{d}x}$. For each of these operators, we consider two different non-homogeneous conditions. As shown in Fig. 6, the results of the predicted and true solution function u coincide exactly in all four cases. Both the domain and boundary loss are of the order of e-5, the same as the baseline (PINN). [Raissi et al., 2019].

For the 1D-Laplace operator, we use the following two functions as the non-homogeneous terms:

$$f(x) = 2 - \sin(x) + 60x - 2((\cos(x))^2 - (\sin(x))^2) \tag{51}$$

$$f(x) = 90(x^8) - (4\pi^2)\sin(2\pi x) - (4\pi^2)\cos(2\pi x) \tag{52}$$

For the 1D operator $\frac{\mathrm{d}^2}{\mathrm{d}x^2} + \frac{\mathrm{d}}{\mathrm{d}x}$, we use the following two functions:

$$f(x) = 3x^2 + 2\pi x\cos(\pi x^2) + \frac{1}{2} + 6x + 2\pi\cos(\pi x^2) - (4\pi x^2)\sin(\pi x^2) \tag{53}$$

$$f(x) = 3x^2 + 6\pi x\cos(3\pi x^2) + \frac{1}{2} + 6x + 6\pi\cos(3\pi x^2) - 36\pi^2 x^2\sin(3\pi x^2) \tag{54}$$

Remark 10.1. Using Monte Carlo based sampling to solve PDEs (as in PINNs) can have undesirable outcomes when dealing with irregular boundary, hence adaptive quadrature methods have been proposed very recently [Omella and Pardo, 2024]. Our method is data-driven does not suffer from such challenges since the quadratures are adaptive by design.

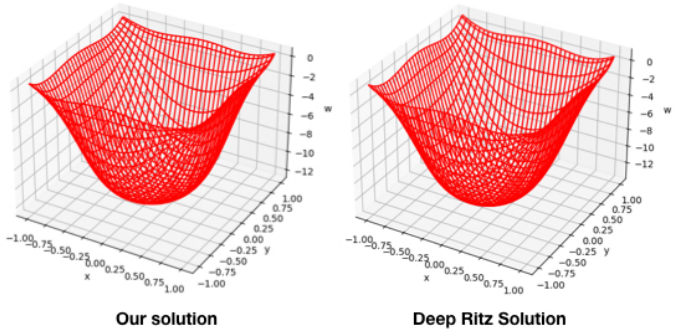


Figure 7: Comparison of solution curves obtained via the proposed learnable method and baseline method of Deep Ritz. Both methods perform equally well.

(B) Numerical experiments with Weak Form. We next apply our learnable quadratures to solve PDEs written in their weak form as described in §3. Here, we consider the following two 1-D operators: (a) 1D-Laplace and (b) $\frac{\mathrm{d}^2}{\mathrm{d}x^2} - \frac{\mathrm{d}}{\mathrm{d}x}$. Similar to the strong form, we present results for two different conditions for each operator. Again, we see from Fig. 8 that, the predicted and true solution function in all cases coincide almost exactly. In terms of the domain and boundary loss, these are of the same order of e^{-3} as the baseline method of hp-VPINN [Kharazmi et al., 2021].

Since the weight functions can be global, in using them as test functions to solve the weak form, we can end up with a global test function. Avoiding this is possible via several schemes: one could either choose a multitude of such test functions or simply use sub-domain splitting as suggested in hp-VPINN over VPINN Kharazmi et al. [2019]. Due to its simplicity, we choose the latter in our experiments.

For the 1D-Laplace operator, we use the following two functions as the non-homogeneous terms:

$$f(x) = 2 - \sin(x) + 60x - 2(\cos^2(x) - \sin^2(x)) \tag{55}$$

$$f(x) = 90(x^8) - 4\pi^2 \sin(2\pi x) - 4 * \pi^2 \cos(2\pi x) \tag{56}$$

For the 1D operator $\frac{\mathrm{d}^2}{\mathrm{d}x^2} - \frac{\mathrm{d}}{\mathrm{d}x}$, we use the following two functions:

$$f(x) = 6x + 2\pi \cos(\pi x^2) - 4\pi x^2 \sin(\pi x^2) - (3x^2 + 2\pi x \cos(\pi x^2) + \frac{1}{2}) \tag{57}$$

$$f(x) = 6x + 6\pi \cos(3\pi x^2) - 36\pi^2 x^2 \sin(3\pi x^2) - (3x^2 + 6\pi x \cos(3\pi x^2) + \frac{1}{2}) \tag{58}$$

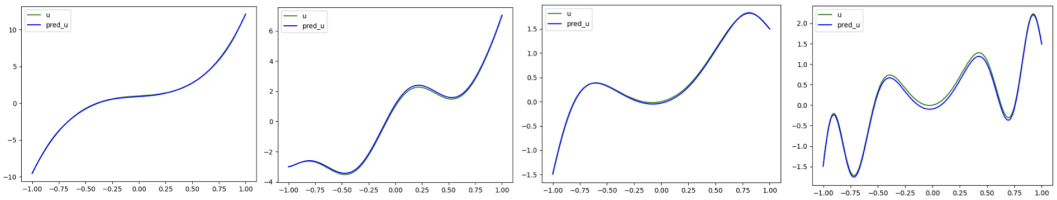


Figure 8: True/Predicted solution functions. 2 right-most two plots for 2 different conditions on the 1D-Laplace operator. 2 left-most two plots for solutions to 2 settings for the operator $\frac{\mathrm{d}^2}{\mathrm{d}x^2} - \frac{\mathrm{d}}{\mathrm{d}x}$

Remark 10.2. For solving PDEs in their strong and weak forms as presented above, we adopt a two stage training scheme. In the first stage, the asymptotic quadrature is learned and in the second stage these learned quadratures are used to either provide orthogonal collocation points in the strong form or test function(s) for the weak form. Our overall procedure is otherwise unchanged.

(C) Energy Method. We now demonstrate the utility of learnable quadrature for solving a PDE where the loss function is derived based on the minimum energy principle.

We consider the 2D-Laplace equation: $\Delta u = -100$ with zero boundary conditions on a square domain: $[-1, 1] \times [-1, 1]$. In the energy form, the loss function has the form

$$L(u) = \int \int_{\Omega} \left(\frac{1}{2} |\Delta u|^2 - fu \right) \mathrm{d}x\mathrm{d}y + \beta \int \int_{\partial\Omega} u^2 \mathrm{d}x\mathrm{d}y \tag{59}$$

where β is a penalty term on the second component denoting the boundary loss. The first component is the loss on the domain. We use our learnable quadrature to approximate both integrals in (59) and compare the solution obtained with the baseline method of Deep-Ritz [Yu et al., 2018] with same number of parameters, running each for roughly 400 epochs. As can be seen from Fig. 7, our method achieves comparable performance, with approximate loss value -2000 in both case.

Remark 10.3. Since our proposed method is, in essence, a data-driven way to sample points, it shows its utility in solving PDEs via *all three* formulations as demonstrated above, where the basic framework remains the same. In the *strong* form, it provides **orthogonal collocation** points. In the *weak form*, it provides **test functions** (which induce the quadrature rules). Finally, in the *energy form* it is used to directly provide a **quadrature rule**.

10.4 Family of PDE via LearnQuad

We specify the details of the family of PDEs which were solved using **LearnQuad** and the procedure outlined in 6.3. The overall algorithm is presented in Algorithm 2.

In all experiments, we used 500 parameters each for the hyper-networks predicting the weight function and solution function as outlined in Section 6.3. Specifically, we used a MLP-based neural network with depth 5; width 100 and $\tanh$ as the activation function. The number of parameters to encode the actual solution function were kept smaller than 20. Using a learning rate of 0.0001, in all cases, the methods took less than 10k epochs to converge. For each family, we sampled 100 instances of the PDE and used a train/test split of $80/20$. We used 600 points as a standard number of points to sample from **LearnQuad**.

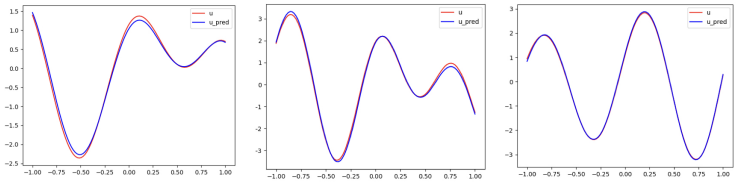
Algorithm 2 Training for a family of PDEs

Input: Operator $\mathcal{L}$; distribution p; parametric form of forcing function F and boundary/initial conditions B. #epochs: T, Training size: n, Learnable modules θ, ϕ; PDE Loss function L and regularization loss l_w (20)
Compute: Generate Training Set, $S = \{\}$
for $i = 1$ **to** $i = n$ **do**
  Sample $\kappa \sim p$
  Get f_κ from F, Get b_κ from B
  S.append$(\kappa, f_\kappa, b_\kappa)$
end for
Compute: Training Loop
for $i = 1$ **to** $i = T$ **do**
  for each $(\kappa, f_\kappa, b_\kappa) \in S$ **do**
    Get w_κ and $\{\tau_0, \tau_1, c_0, d_0, d_1\}_\kappa$ from $\theta(\kappa)$
    Get u_κ from $\phi(\kappa)$
    Use §5 to get quadrature nodes $\{x_l\}_\kappa$
    Loss:$l = L(\{\mathcal{L}u_\kappa(x_l)\}_\kappa, \{f_\kappa(x_i)\}_\kappa) + l_w$
    Gradient based update for θ and ϕ based on l
  end for
end for
Output: Learned modules θ and ϕ

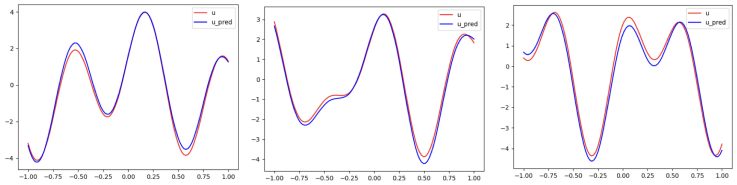
10.4.1 Family of Laplace Equation

Here, we consider the 1D-Laplace operator which has the following parametric representation for the non-homogeneous function:

$$f_\kappa(x) = -(a\pi^2\mu^2 \sin(\pi\mu x) + b\pi^2\nu^2 \cos(\pi\nu x)) \tag{60}$$



(a) Family corresponding to $p \sim U(1, 2.5)$



(b) Family corresponding to $p \sim U(1, 4.0)$

Figure 9: Test results for solving PDEs from two different family each with 1-D Laplace operator. The second family 9(b) has more variance than the first. In all cases, the predicted solution almost coincides with the original solution

where, $\kappa = \{a, b, \mu, \nu\}$ belong to different distribution. In our experiments, we choose these distributions as uniform, but our method can handle any distribution. In Figure 9 we show the performance on the test set. It can seen that the predicted solution is very close to the true solution.

10.4.2 Family of Heat Equation

We consider the one dimensional heat equation and sample the heat diffusivity, c; initial distribution, f; and two boundary conditions, T_l and T_r. The PDE along with initial and Dirichlet boundary conditions is given as follows:

$$\frac{\partial u}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \ t \in [0, 2] \tag{61}$$

$$u(-1, t) = T_l, \quad t \in [0, 2] \tag{62}$$

$$u(1, t) = T_r, \quad t \in [0, 2] \tag{63}$$

$$u(x, 0) = f(x), \quad x \in [-1, 1] \tag{64}$$

We perform experiments, with two choices for the initial distribution:

$$f(x) = mx + n \tag{65}$$

$$f(x) = a \sin(\pi\theta x) + b \cos(\pi\phi x) \tag{66}$$

We present a visualization of the true (numerical) solution obtained using the same number of domain points as LearnQuad, the predicted solution and their relative error in Figure 10. The parameters were sampled from uniform distribution $U(0.8, 2.0)$.

10.4.3 Family of Wave Equation

We consider the 1D wave equation and sample the wave speed, c and the initial position, f and velocity, g. The PDE along with initial conditions is given below:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \ t \in [0, 2] \tag{67}$$

$$u(x, 0) = f(x), \quad x \in [-1, 1] \tag{68}$$

$$\frac{\partial u}{\partial t}(x, 0) = g(x), \quad x \in [-1, 1] \tag{69}$$

We perform experiments, with the following two sets of initial conditions:

$$f(x) = mx, \quad g(x) = a + x; \quad x \in [-1, 1] \tag{70}$$

$$f(x) = mx + n, \quad g(x) = a \sin(\pi\theta x) + b \cos(\pi\phi x); \quad x \in [-1, 1] \tag{71}$$

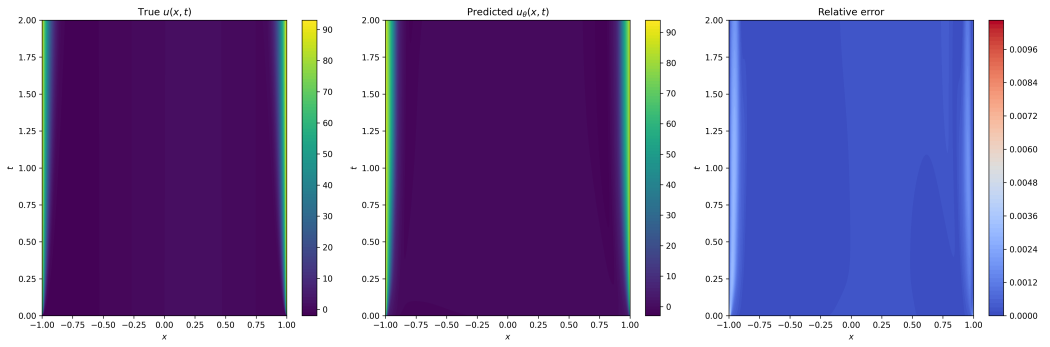


Figure 10: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of heat equation using (61)-(64) and (66)

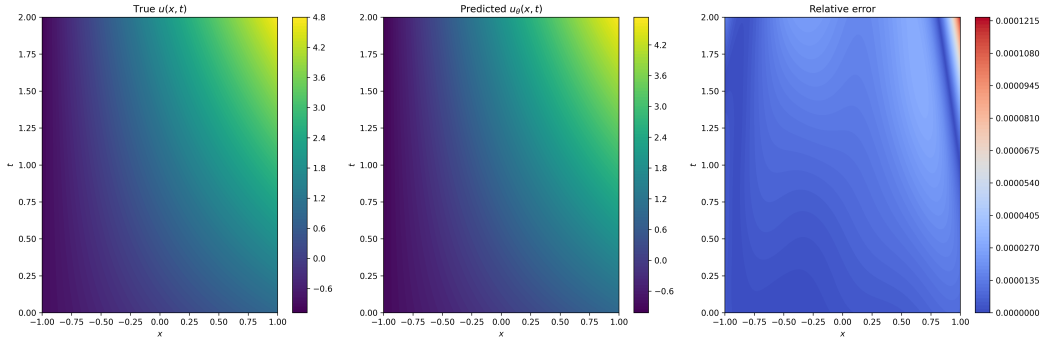


Figure 11: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of wave equation using (67)-(69) and (70)

We present visualization of the true (numerical) solution obtained using the same number of domain points as LearnQuad, the predicted solution and their relative error in Figure 11 and Figure 12. The parameters were sampled from uniform distribution $U(0.5, 3.0)$.

10.4.4 Family of Advection Equation

We consider the one dimensional advection equation and sample the advection speed, c and the initial position f. The PDE along with the initial conditions is given by:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \quad x \in [-1, 1],\ t \in [0, 2] \tag{72}$$

$$u(x, 0) = f(x), \quad x \in [-1, 1] \tag{73}$$

We conduct experiments with the two following choices for the initial displacement:

$$f(x) = mx + n, \quad x \in [-1, 1] \tag{74}$$

$$f(x) = a\sin(\pi\theta x) + b\cos(\pi\phi x), \quad x \in [-1, 1] \tag{75}$$

We present visualization of the true (numerical) solution obtained using the same number of domain points as LearnQuad, the predicted solution and their relative error in Figure 13 and Figure 14. The parameters are sampled from the uniform distribution $U(0.5, 1.5)$.

10.4.5 Family of Burger's Equation

We consider the one dimensional viscous Burgers' equation which is a non-linear PDE. We sample the diffusivity coefficient, c; initial velocity distribution, f; and the two boundary conditions, T_l and

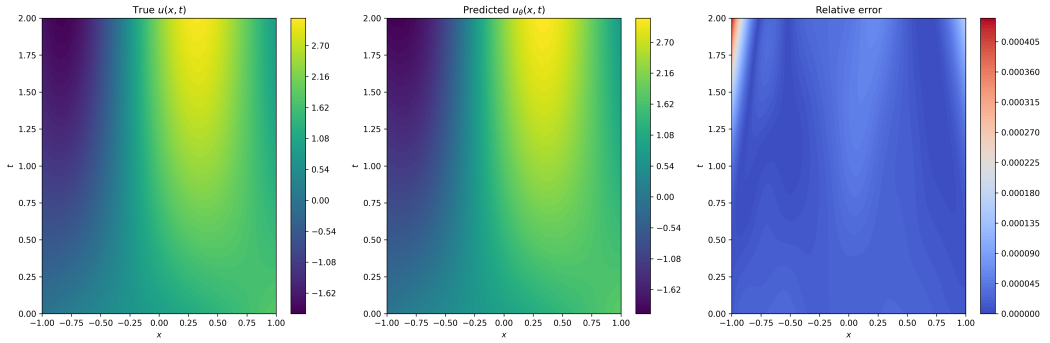


Figure 12: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of wave equation using (67)-(69) and (71)

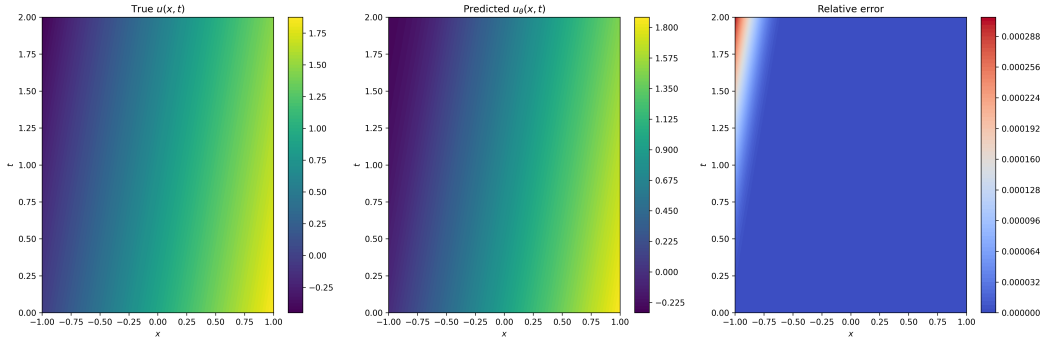


Figure 13: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of advection equation using (72)-(73) and (74)

T_r. The PDE along with the initial and boundary conditions is given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = c^2\frac{\partial^u}{\partial x^2}, \quad x \in [-1,1],\ t \in [0,2] \tag{76}$$

$$u(-1,t) = T_l, \quad t \in [0,2] \tag{77}$$

$$u(1,t) = T_r, \quad t \in [0,2] \tag{78}$$

$$u(x,0) = f(x), \quad x \in [-1,1] \tag{79}$$

We consider the following two different choices for the initial condition:

$$f(x) = m, \quad x \in [-1,1] \tag{80}$$

$$f(x) = a\exp^{-bx^2}, \quad x \in [-1,1] \tag{81}$$

We present visualization of the true (numerical) solution obtained using the same number of domain points as LearnQuad, the predicted solution and their relative error in Figure 15, Figure 16 and Figure 17. The parameters are sampled from the uniform distribution $U(0.01, 0.1)$.

11 Additional Discussions

11.1 Parallel computation of Large number of nodes

LearnQuad's design, specifically our choice of parameterization avoids explicit computation of several contour integrals, allows for significant parallelization. This is due to two reasons: (a) using learnable linear layers for modulating function and expansion coefficients allows bypassing compute-intensive steps such as contour integral (b) computing each quadrature node via an independent root-finding procedure. This is crucial since it means that the root-finding process for all nodes and the neural

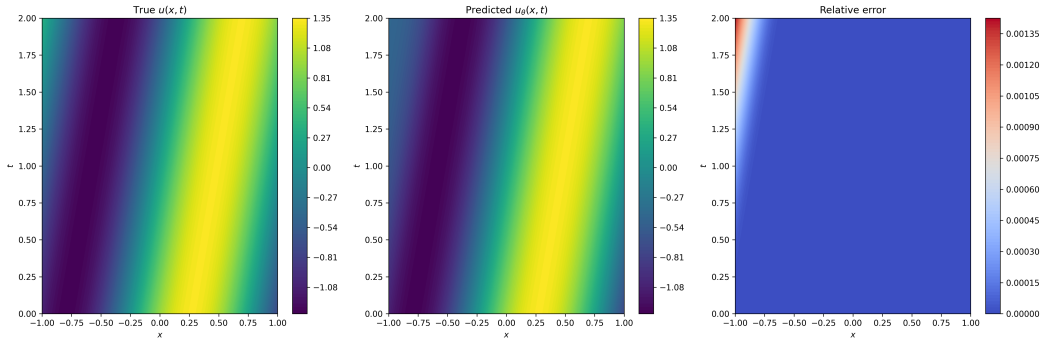


Figure 14: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of advection equation using (72)-(73) and (75)

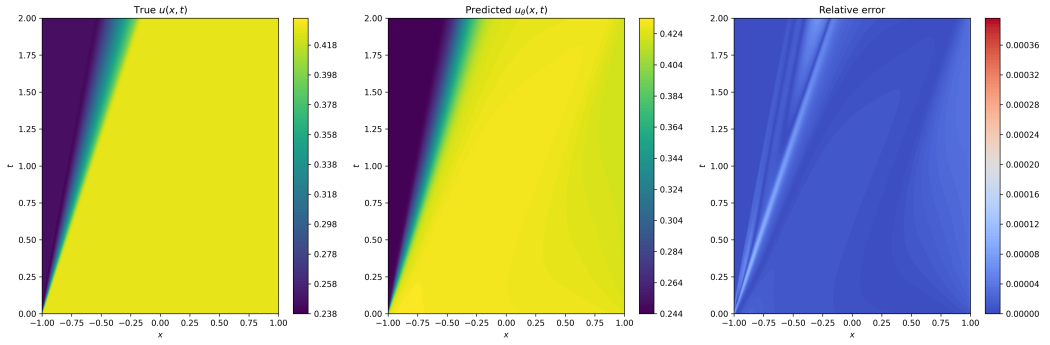


Figure 15: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of viscous Burgers' advection equation using (76)-(79) and (80)

network evaluations to get the parameters can be performed concurrently, and maps well to GPUs. We checked wall-clock time by varying the number of nodes on our commodity GPU (NVIDIA 2080 Ti).

Table 8: Generation time for different numbers of nodes.

Number of Nodes	**Generation Time (sec)**
10	0.0008
100	0.0009
1000	0.0009
10000	0.0012
100000	0.0098
1000000	0.0754

We see from Table 8 that the runtime remains roughly flat upto 10000 nodes. Beyond this we see a modest sub-linear increase in runtime. This increase is due to practical/implementation limitations. As number of nodes reaches millions, transferring tensors associated with nodes between compute units and memory increases runtime. This memory bandwidth issue specific to our GPU can be mitigated (although not eliminated) by higher-end hardware and/or optimized implementation. The core algorithm, by design, remains fully parallel.

11.2 Computational Requirement for LearnQuad

We present wall-clock time per iteration of the baseline methods and LearnQuad below for the convection PDE. We used R3's Pytorch code [Daw et al., 2023] for all methods, explaining longer runtimes vs. our JAX version. We also tested PINN in JAX for transparency. LearnQuad's higher

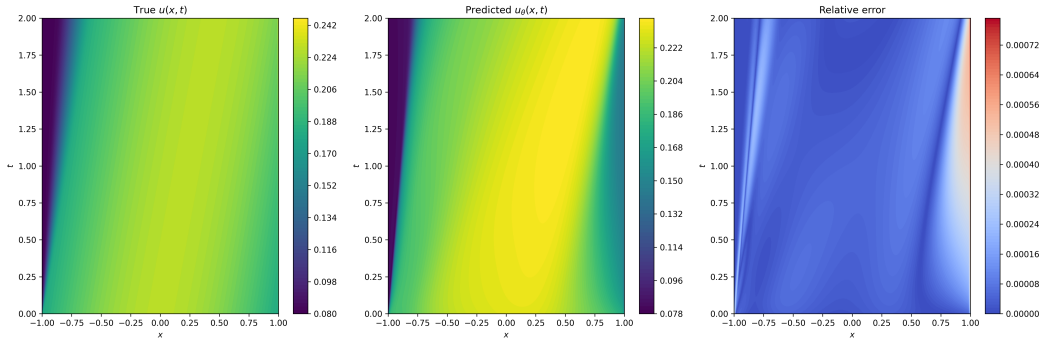


Figure 16: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of viscous Burgers' equation using (76)-(79) and (81)

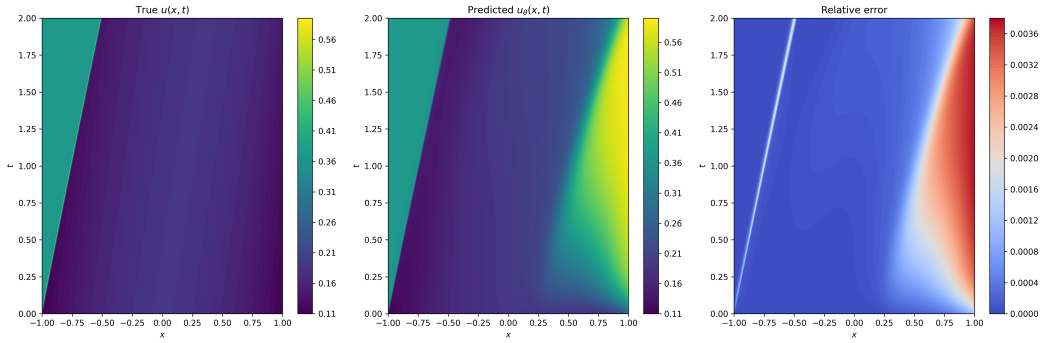


Figure 17: (Left)True solution, (Center) Predicted solution and (Right) Relative error for an instance from the test set of the family of viscous Burgers' equation using (76)-(79) and (81)

runtime over efficient JAX implementation of PINN is likely due to the extra learnable model that is serialized before the PINN model. We find the trade-off fair given better performance. Our model converges faster than R3 and other baselines.

11.3 PINNs and Classical Solvers

Our proposed method is not designed to compete with classical solvers. While for simple problems, classical methods are indeed effective, one motivation behind the sizable PINN literature is an alternative which is advantageous in many scenarios: (a) PINN based solutions are mesh-independent; (b) they rely on automatic-differentiation which are easier to implement; and (c) can handle non-linearity effectively given the universal function approximation properties of neural networks. We note that PINN based methods never use "classical solution" as the ground truth in the training procedure at all. It is only used to evaluate a test time performance metric. This is needed in cases where the PDE solution is not given in a closed form, which is true for most scenarios.

11.4 PINN loss over PDE solver

Our object of interest in this work is PINN. PINNs provide a mesh independent solution, are more amenable to non-linearities and are easier to scale and implement. Hence, PINNs offer many benefits in several cases and for this reason, are being studied extensively. Next, we justify our choice of PINN loss instead of a PDE solver.

While the learnable quadrature rule is amenable for classical solvers, there are several issues. Suppose that we use a classical solver instead of a PINN loss. This means that for each update of parameters θ in learnable weight function w_θ (which induces the quadrature) we will need to (i) generate quadrature points using current w_θ, (ii) solve the system of equations (either implicitly using a solver

Table 9: Average time per iteration for different methods.

Method	Avg. Time per Iteration (sec)
LearnQuad	0.00461 (Jax)
R3	0.01172
PINN	0.00274 (Jax)
Causal R3	0.01444
PINN (fixed)	0.01291
PINN (uniform)	0.01261
Curriculum Regularization	0.01278
Causal PINN	0.01317
RAR-G	0.01305
RAR-D	0.01316
RAD	0.01342
L-inf	0.01337

or iteratively), (iii) compute some loss/quality and (iv) update θ to improve this metric. This poses several challenges. Explicit (iterative) solvers are memory-intensive when unrolling across time steps, sensitive to numerical instabilities, thereby requiring fine time steps and increased computational cost. When differentiating through a numerically unstable solver, the gradients can become inaccurate or blow up. Implicit solvers demand solving linear or nonlinear systems. Computing Jacobians for implicit differentiation requires significant computational resources. Furthermore, matrix inversion or solving linear systems as part of implicit differentiation introduces high computational overhead. Hence, we can agree that integrating PDE solvers into neural network modules presents challenges for both explicit and implicit solvers due to the above mentioned issues in computing gradients which are necessary to update the models via back-propagation. Therefore, in order to make learnable quadrature feasible – the main goal of this work – we leverage the PINN loss which is more suited for the end-to-end learning framework.

Another aspect worth mentioning is regarding the setup for a family of PDEs. Without a scheme to learn the common structure shared between different instances of the PDE, it would require solving each instance separately at each desired resolution. To conclude, the choice of PINN loss is important not only for solving individual PDEs, but also for permitting the learning of quadrature rules that can then be used across multiple problems/solution schemes.

11.5 Contrast with other adaptive methods for PINNs

Our main contribution is not just solving PDEs, but learning how to optimally sample points based on the PDE's structure. We emphasize that advantages stem directly from our core theoretical contribution: the learnable weight function that induces problem-specific quadrature rules. This is fundamentally different from both classical adaptive methods and other existing ML approaches like R3Daw et al. [2023], RARLu et al. [2021], RADWu et al. [2023]; all of whom invariably rely on computing error estimates through residual-based estimators or gradient thresh-holding which are problem-specific, need to be chosen carefully, and sometimes may need to solve additional local problems. Instead we adaptively learn where refinement may be needed in an end-to-end fashion in conjunction with the PINN loss and no additional explicit error estimation is required.