BWLER : Barycentric Weight Layer Elucidates a Precision-Conditioning Tradeoff for PINNs

Jerry Liu[1]* **Yasa Baig**[2] **Denise Hui Jean Lee**[3] **Rajat Vadiraj Dwaraknath**[1]
Atri Rudra[4] **Chris Ré**[3]

[1]Institute for Computational & Mathematical Engineering, Stanford University
[2]Department of Bioengineering, Stanford University
[3]Department of Computer Science, Stanford University
[4]Department of Computer Science & Engineering, University at Buffalo

Abstract

Physics-informed neural networks (PINNs) offer a flexible way to solve partial differential equations (PDEs) with machine learning, yet they still fall well short of the machine-precision accuracy many scientific tasks demand. This motivates an investigation into whether the precision ceiling comes from the ill-conditioning of the PDEs themselves or from the typical multi-layer perceptron (MLP) architecture. We introduce the **Barycentric Weight Layer (BWLER)**, which models the PDE solution through barycentric polynomial interpolation. A BWLER can be added on top of an existing MLP (a BWLER-hat) or replace it completely (explicit BWLER), cleanly separating how we *represent* the solution from how we take its *derivatives* for the physics loss. Using BWLER, we identify fundamental precision limitations within the MLP: on a simple 1-D interpolation task, even MLPs with $O(10^5)$ parameters stall around 10^{-8} relative error – about eight orders above `float64` machine precision – before any PDE terms are added. In PDE learning, adding a BWLER lifts this ceiling and exposes a tradeoff between achievable accuracy and the conditioning of the PDE loss. For linear PDEs we fully characterize this tradeoff with an explicit error decomposition and navigate it during training with spectral derivatives and preconditioning. Across five benchmark PDEs, adding a BWLER on top of an MLP improves ℓ_2 relative error by up to $30\times$ for convection, $10\times$ for reaction, and $1800\times$ for wave equations while remaining compatible with first-order optimizers. Replacing the MLP entirely lets an explicit BWLER reach near-machine-precision on convection, reaction, and wave problems (up to *10 billion times better* than prior results) and match the performance of standard PINNs on stiff Burgers' and irregular-geometry Poisson problems. Together, these findings point to a practical path for combining the flexibility of PINNs with the precision of classical spectral solvers.

1 Introduction

Partial differential equations (PDEs) are the standard tool for modeling complex phenomena across science and engineering [12]. Traditionally, PDEs have been solved using numerical methods (e.g. finite element or spectral methods [17, 4]) but there has been recent interest in leveraging modern machine learning (ML) techniques to solve these classical problems [6, 20]. Producing better ML-based methodologies for PDEs could enable faster simulations while maintaining the high fidelity of traditional numerical methods, with applications from weather forecasting to design optimization [25, 7].

Physics-informed neural networks (PINNs) [29] parametrize the solution of a PDE with a multi-layer perceptron (MLP) and enforce PDE constraints with least-squares losses during training. The main benefit of this *physics-informed* framework is flexibility: it requires no meshing, handles irregular geometries

*Corresponding author: `jwl50@stanford.edu`

Preprint. Under review.

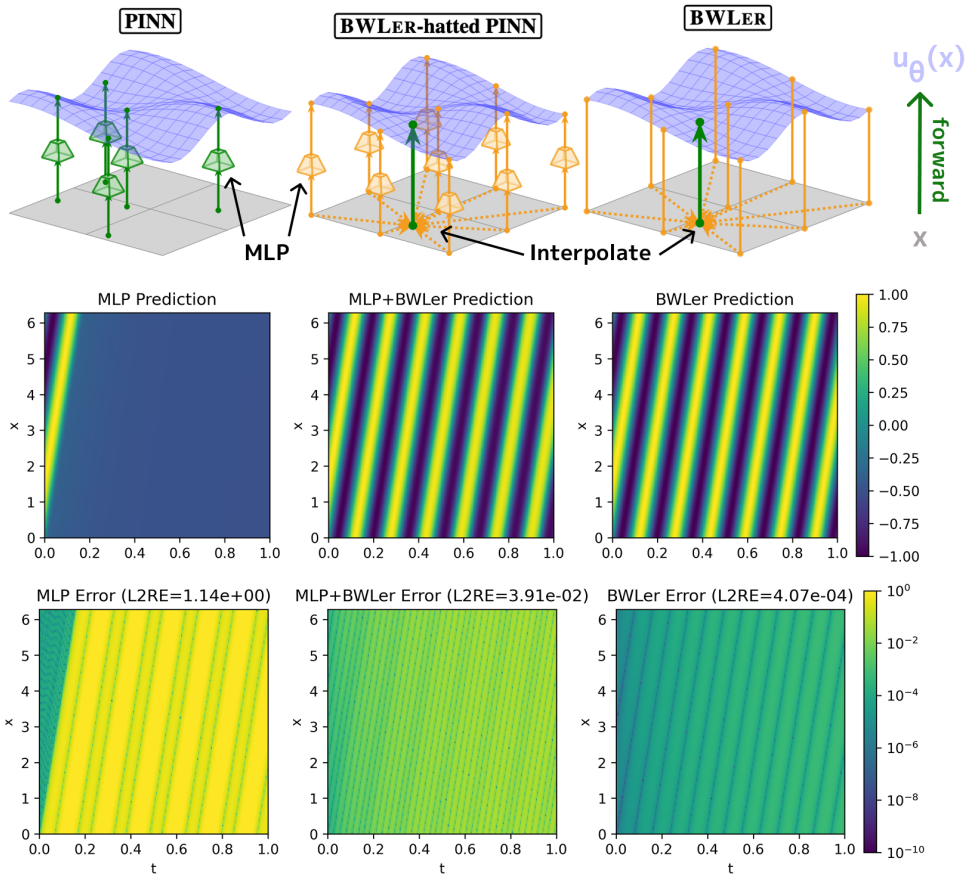

Figure 1: **Top: model architecture comparison.** Standard PINN evaluates an MLP throughout the domain (left). BWLER interpolates globally based on values at discrete grid nodes; BWLER-hatted MLP obtains values using an MLP (middle), explicit BWLER parameterizes values directly (right). **Bottom: results for convection equation** [31]. Standard PINN stagnates at a suboptimal local minimum (left); BWLER-hatted MLP finds a qualitatively correct solution (middle); explicit BWLER converges to higher precision (right).

gracefully, and provides a unified methodology for diverse PDE types [30, 20, 28]. However, PINNs have struggled to achieve high-precision solutions [27, 26, 24], crucial for scientific applications such as turbulence modeling or maintaining stable temporal rollouts [15]. PDEs are particularly challenging because of their fundamentally ill-conditioned differential operators; despite recent progress investigating the precision saturation of PINNs on PDE problems [36, 22, 38, 31], it remains unclear to what extent the issues stem from problem-inherent ill-conditioning versus the models' parameterizations.

In this work, we aim to disentangle and analyze the sources of precision limitations in PINNs. Specifically, we ask: *(i) are there inherent precision bottlenecks in the MLP architectures used by PINNs, and (ii) how does the difficulty of the underlying PDE affect the precision that can be achieved?* Our study has the following three parts:

- **We identify fundamental MLP precision limitations in a simple setting.** Through systematic experiments on 1-D interpolation, we show that MLP precision plateaus around 10^{-8} ℓ_2 relative error (L2RE). This is roughly eight orders of magnitude above `float64`'s machine epsilon ($2^{-52} \approx 2.22 \times 10^{-16}$), even without PDE constraints. We demonstrate that this limitation persists as we scale network width by $16\times$ and depth by $4\times$, with precision improving by just 1–2 orders of magnitude even with $1000\times$ more parameters (Figure 2). In contrast, classical polynomial representations with just 10–100 parameters can provably achieve machine precision (Theorem 2.2). Our results point to precision bottlenecks stemming from the neural network parameterization itself even beyond optimization challenges specific to PDEs.

- **We propose a barycentric interpolation framework for PDE learning.** Motivated by the precision of polynomials, we introduce BWLER[1], a simple baseline that can be used as a drop-in replacement for standard PINN architectures. BWLER parameterizes the solution function as a *barycentric polynomial interpolant* [33, 3], where the model is defined by the function values it takes on a pre-specified discrete grid in the domain (Figure 1, top; Algorithm 1). Our method builds upon decades of work using polynomial interpolants to solve numerical PDEs [4, 14, 32, 8, 1]; BWLER effectively embeds a pseudo-spectral solver into the physics-informed framework while leveraging auto-differentiation and ML optimizers. Excitingly, BWLER lets us decouple our choice of model parameterization (e.g. explicit grid, neural network) from our PDE derivatives calculations (e.g. finite differences, spectral derivatives). Using BWLER, we next ablate the MLP to study the effect of model parameterization vs. the PDE ill-conditioning on precision.
- **We characterize a precision-conditioning tradeoff with BWLER.** We investigate two variants of BWLER (Algorithm 1). **BWLER-hatted MLPs**, which apply BWLER atop existing MLPs, outperform standard MLPs on convection, reaction, and wave equation benchmark problems by $30\times$, $10\times$, and $1800\times$ respectively. We find BWLER improves the PDE loss conditioning, decreasing mean eigenvalue by 5–$10\times$ (Figure 8, Figure 9). We then turn to **explicit BWLER**, where the model is directly parameterized by its function values on the pre-specified grid. We fully characterize the training error of explicit BWLER on linear PDEs, identifying a fundamental tradeoff between BWLER's maximum achievable precision and the conditioning of the optimization problem (Theorem 5.1). Motivated by our error decomposition, we vary preconditioning and derivative computations to navigate the tradeoff space during training. For the first time, we achieve near *machine precision* with PINNs on convection, reaction, and wave equation benchmarks (up to *10 billion times better L2RE* than prior PINN methods) and match state-of-the-art performance on Burgers' and irregular-geometry Poisson problems (Table 2).

2 Background

We provide background information on physics-informed neural networks and barycentric Lagrange interpolation. We defer a lengthier discussion of related work to Appendix A.

2.1 Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) [30] propose a flexible and general framework to solve PDEs using neural networks, capable of treating a variety of boundary conditions and geometries. Consider a PDE of the form:

$$\begin{cases} \mathcal{F}(u,x)=0, & x\in\Omega_{\text{PDE}} \\ u(x)=u_0(x), & x\in\Omega_{\text{IBC}} \end{cases} \tag{1}$$

where $\mathcal{F}$ is a differential operator, Ω_{PDE} is the domain, and $\Omega_{\text{IBC}}\subset\Omega_{\text{PDE}}$ denotes the initial condition region. The PINN framework represents the solution to Equation (1) as a parametric model u_θ and formulates a composite loss function combining a physics term and a boundary term:

$$\mathcal{L}(u)=\mathcal{L}_{\text{PDE}}(u)+\lambda_{\text{IBC}}\mathcal{L}_{\text{IBC}}(u), \tag{2a}$$

$$\mathcal{L}_{\text{PDE}}(u)=\mathbb{E}_{x\in\Omega_{\text{PDE}}}\left[(\mathcal{F}(u,x))^2\right], \tag{2b}$$

$$\mathcal{L}_{\text{IBC}}(u)=\mathbb{E}_{x\in\Omega_{\text{IBC}}}\left[(u(x)-u_0(x))^2\right] \tag{2c}$$

This framework requires the following operations from its model class: (i) **Evaluation**, computing $u_\theta(x)$ for any $x\in\Omega_{\text{PDE}}$, and (ii) **Differentiation**, computing partial derivatives $\partial^{(k)}u_\theta/\partial x_i^{(k)}(x)$ for any $x\in\Omega_{\text{PDE}}$. In order to leverage auto-differentiation during optimization, both operations must be differentiable with respect to model parameters θ. Although any models satisfying these properties can be used for physics-informed learning (*e.g.* Gaussian processes [29]), recent work focuses on neural networks [30].

2.2 Barycentric interpolants and spectral methods

Barycentric Lagrange interpolation is a classical technique for polynomial-based function approximation, specified entirely by the function's values at a set of interpolation nodes [3].

[1] Code available at `https://github.com/HazyResearch/bwler`.

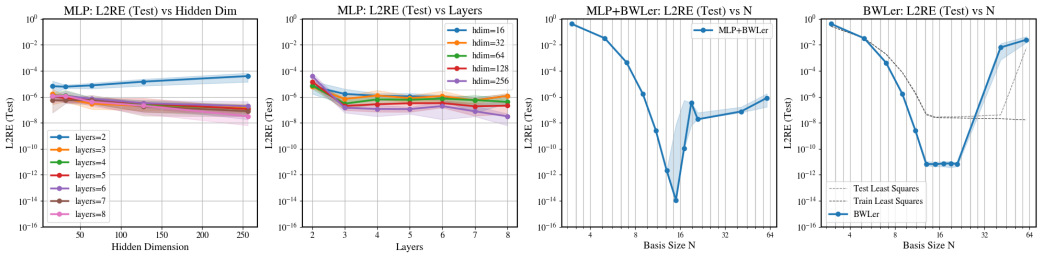

Figure 2: Left: MLPs struggle to interpolate 1-D functions beyond 10^{-8} MSE (pictured: $f(x) = \sin(4x)$), even as we scale model width and depth. Right: BWLER achieves spectral accuracy (10^{-12} MSE); BWLER-hat improves MLP's MSE by more than $100{,}000\times$. Least squares Chebyshev interpolation (fit on train, evaluated on test) is also reported (right).

Definition 2.1. Given $N+1$ distinct nodes $\{x_j\}$ and values $f(x_j)$, the *barycentric Lagrange interpolant* is:[2]

$$p_N(x) = \frac{\sum_{j=0}^{N} \frac{w_j}{x-x_j} f(x_j)}{\sum_{j=0}^{N} \frac{w_j}{x-x_j}}, \tag{3}$$

where $\{w_j\}$ are *barycentric weights*: $w_j = 1/\left(\prod_{k\neq j}(x_j - x_k)\right)$.

Derivatives can be efficiently computed using differentiation matrices or FFT-based methods, depending on the node distribution [32]. See Section 4 and Appendix B for more details.

For well-chosen nodes (*e.g.* Chebyshev-distributed [33]) and smooth functions, the resulting interpolants exhibit spectral convergence – *exponentially decaying error* for the function and its derivatives.

Theorem 2.2 (Chebyshev interpolants exhibit spectral convergence [33, 4])**.** *Let $f : [-1,1] \to \mathbb{R}$ extend to an analytic function on a Bernstein ellipse E_ρ with foci at ± 1 and sum of semiaxes $\rho > 1$. Let p_n be the degree-n Chebyshev interpolant of f. Then:*

$$\|f - p_n\|_\infty \leq \frac{4M\rho^{-n}}{\rho - 1}, \qquad \|f^{(k)} - p_n^{(k)}\|_\infty \leq \frac{C_k M \rho^{-n}}{(\rho-1)^{k+1}},$$

for some constant C_k depending on k and ρ, where $M = \max_{z\in E_\rho} |f(z)|$.

Barycentric interpolation forms the foundation of classical pseudo-spectral methods for solving numerical PDEs [4, 32, 8], where function values on a fixed grid are used to compute derivatives spectrally. This approach underlies well-established numerical solvers (e.g., Chebfun [1]), and provides a principled framework for high-precision computation. Our method, BWLER, builds on this line of work, adapting it for use with gradient-based optimization and machine learning.

3 Neural networks struggle with precise 1-D interpolation

To disentangle the effects of PDE conditioning from model parameterization, we begin with a simplified setting: one-dimensional smooth function interpolation. This lets us isolate the approximation behavior of different model classes in a well-conditioned regime. Surprisingly, we find that MLPs consistently plateau in precision and scale poorly with larger models (Section 3.2). In contrast, polynomial interpolation admits a complete theoretical analysis and provably converges to (near)-machine precision (Section 3.3).

3.1 Experimental setup

We study the task of one-dimensional function interpolation on the domain $[-1, 1]$, isolating approximation behavior from PDE constraints. We evaluate on sinusoids of the form $\sin(kx)$ for frequencies $k \in \{1,2,4,8,16,32\}$. See Appendix C.1 for more details.

[2] Although Equation (3) is a rational function with poles at the interpolation nodes, the barycentric form is numerically stable even for large N, and avoids the catastrophic cancellation associated with the standard Lagrange formula [3, 1].

For each target function, we generate a training set of $N_{\text{train}} = 100$ points sampled uniformly at random from the domain, and evaluate performance on a dense test grid of $N_{\text{test}} = 1000$ points. We report relative ℓ_2 error (L2RE) on the test grid (Appendix B).

3.2 MLPs exhibit precision bottlenecks

We use a fully-connected MLP with $\tanh$ activations, a standard architecture in prior work on PINNs [20, 16, 37]. Given a set of training points $\{(x_i, f(x_i))\}_{i=1}^{N_{\text{train}}}$, the model is trained to minimize the mean squared error (MSE). We use the Adam optimizer [21] with a learning rate of 10^{-3} and a cosine decay schedule. To study the effect of model capacity, we sweep network widths within $\{2^4,...,2^8\}$ and depths from 2–8 layers ($16\times$ and $4\times$ ranges, respectively). We also sweep across different levels of function smoothness.

Figure 2 (left) shows representative results for $f(x) = \sin(4x)$; comprehensive results are provided in Appendix C.1. We find that MLPs consistently stagnate well above machine epsilon – for the function shown, relative error plateaus around 10^{-8} – 8 orders of magnitude worse than `float64`'s machine precision of 2.22×10^{-16}. Moreover, precision scales poorly with model size: even when increasing the number of parameters by over $1000\times$ ($400{,}000$ parameters for the largest MLP we consider), we observe only a 10–$100\times$ improvement in accuracy. These results suggest that the MLP architecture itself imposes a fundamental bottleneck on achievable precision.

3.3 Polynomials achieve exponential convergence

In our experiments, we use an N-element Chebyshev polynomial basis and solve for the optimal coefficients via least squares on the training set. This reduces to solving a linear system $A\mathbf{c} = \mathbf{f}$, where A is the matrix of Chebyshev basis functions evaluated at the training nodes, and $\mathbf{f}$ contains the target function values at those nodes. More details about the polynomial baseline are in Appendix C.1.3.

Figure 2 (right, dotted) shows the empirical error decay of polynomial interpolation on the same target used in the MLP experiment. As predicted by theory (Theorem 2.2), the error decays exponentially in N. In particular, for the optimal basis size, the polynomial baseline achieves relative errors near machine epsilon – up to $10{,}000\times$ better L2RE than the MLP with just 20–50 basis functions. These results motivate the BWLER architecture we introduce in the next section.

4 BWLER: a simple baseline using barycentric interpolants

BWLER proposes *barycentric interpolants* as a drop-in replacement for MLPs in the physics-informed framework of PINNs. For clarity, we present our method in the 1-D setting over the domain $\Omega = [-1,1]$, though the approach directly generalizes to periodic domains (using trigonometric instead of Chebyshev polynomials) and higher dimensions (via tensor products). See Appendix B for details.

4.1 Model parameterization

Let $\{x_j\}_{j=0}^{N}$ denote the Chebyshev nodes of the second kind (Chebyshev-Gauss-Lobatto points [32]) defined by $x_j = \cos(j\pi/N), j = 0,...,N$. Our model is defined as the unique polynomial f_θ that interpolates the points

$$(x_0, f_\theta(x_0)),...,(x_N, f_\theta(x_N)),$$

where we specify our model via the values $\{f_\theta(x_j)\}$ it takes on the discrete set of Chebyshev nodes[3].

We consider two possible parameterizations of these node values:

- **Explicit**. Each value is treated as its own, independently trainable parameter, meaning the full set of trainable parameters in the model is $\theta = [\theta_0,...,\theta_N]^\top$, where $\theta_j := f(x_j)$.
- **Implicit**. Like standard PINNs, these define an MLP that specifies function values at discrete node locations. Unlike standard PINNs, the MLP is only evaluated at these node locations, and

[3] BWLER effectively parameterizes the Lagrange interpolant in *value space*, i.e., directly in terms of the function values at interpolation nodes, rather than in *coefficient space* as in classical polynomial bases. This avoids the instability and ill-conditioning associated with solving for global polynomial coefficients [3].

barycentric interpolation is used to define function values over the full domain. We also term this a **BWLER-hatted MLP**.

Standard PINN	BWLER-hatted PINN
function EVALUATE(x, θ)	**function** EVALUATE(x, θ)
return $MLP_\theta(x)$ ▷ forward pass on x	$f_j \leftarrow \text{MLP}_\theta(x_j)$ ▷ forward pass on grid
function DIFFERENTIATE(x, θ, k)	**return** $\text{BaryInterp}(x, \{f_j\})$ (Algorithm 2)
return $\frac{\partial^k u_\theta(x)}{\partial x^k}$ ▷ autodiff	**function** DIFFERENTIATE(x, θ, k)
	$f_j \leftarrow \text{MLP}_\theta(x_j)$ ▷ forward pass on grid
	$d_j^{(k)} \leftarrow \text{SpectralDeriv}(\{f_j\}, k)$ (Algorithm 3)
	return $\text{BaryInterp}(x, \{d_j^{(k)}\})$ (Algorithm 2)

Algorithm 1: Evaluation and differentiation operations for a standard physics-informed neural network (left) versus a BWLER-hatted neural network (right) – differences in blue.

As required for the physics-informed framework (see Section 2.1), our model has efficient and auto-differentiable implementations of both *evaluation* and *differentiation* operations:

Evaluation. Given the node values $\{f_\theta(x_j)\}_{j=0}^N$, we compute the interpolant $f_\theta(x)$ at any point $x \in \Omega$ using the barycentric formula, Equation (3), where the barycentric weights for Chebyshev–Gauss–Lobatto nodes are $w_j = ((-1)^j)/(1+\delta_{j0}+\delta_{jN})$.

Differentiation. Derivatives are computed efficiently via the Discrete Cosine Transform (DCT). Given node values $\{f_\theta(x_j)\}_{j=0}^N$, the differentiation operation involves transforming to frequency space via DCT, applying differentiation in frequency space (a diagonal operator), and transforming back to physical space via inverse DCT [4]. This yields the derivative values $\mathbf{f}' = [f'(x_0), \ldots, f'(x_N)]^\top$ at the Chebyshev nodes in $O(N \log N)$ operations. To compute the derivative at any point $x \in \Omega$, we apply the barycentric interpolation formula (Equation (3)), plugging in the derivative $\mathbf{f}'$ as the node values. Higher-order derivatives can be obtained by repeating this process. Detailed descriptions of the evaluation and differentiation operations are provided in Appendix B, including pseudocode implementations (Algorithms 2, 3).

4.2 BWLER achieves exponential convergence on interpolation

In Figure 2 and Appendix C.1, we empirically evaluate BWLER in the 1-D interpolation setting, comparing both explicit BWLER models and BWLER-hatted MLPs (where BWLER acts as a final layer atop a standard MLP). We find that explicit BWLER, trained with Adam, closely follows the exponential convergence behavior of the polynomial baseline from Section 3.3. Moreover, with proper choice of N, BWLER-hatted MLPs substantially outperform standard MLPs on smooth targets, improving L2RE by up to $100{,}000\times$ (Appendix C.1.2).

In this setting, we can fully characterize the error convergence of BWLER; we prove that explicit BWLER achieves exponentially decaying test error and convergence under gradient descent:

Theorem 4.1 (Exponential convergence of BWLER on interpolation, informal)**.** *We approximate an analytic function f by fitting an $(N+1)$-parameter* BWLER *for t steps of gradient descent on M sample points in $[-1,1]$. Then its sup-norm error decomposes into*

$$\|f - f_N^{(t)}\|_\infty \leq \underbrace{O(\rho^{-N})}_{\textit{expressivity gap}} + \underbrace{\widetilde{O}\left(e^{-t/\kappa^2}\right)}_{\textit{optimization gap}}, \tag{4}$$

where $\rho > 1$ only depends on the function smoothness, and κ is the interpolation matrix's condition number.

In Equation (4), the expressivity gap is unavoidable and comes from the standard Cheybshev approximation bound (Theorem 2.2) and the optimization gap comes from gradient descent's convergence on least squares [5]. Intuitively, after we choose N large enough that BWLER can express

the target function up to error ϵ, gradient descent will then converge exponentially to it in $O(\log(1/\epsilon))$ steps. See Theorem D.4 for the precise theorem statement and Appendix D.1 for the proof.

5 Physics-informed BWLER and the precision-conditioning tradeoff

We evaluate BWLER, first implicit (BWLER-hatted MLP), then explicit, on benchmark PDE problems [31, 16], including linear (convection, wave), nonlinear (reaction), stiff (Burgers'), and irregular-domain problems (Poisson). In doing so, we aim to disentangle the effects of model architecture and PDE conditioning on precision and optimization behavior. Our key result is Theorem 5.1, where we fully characterize the tradeoffs between maximum achievable precision and training convergence for explicit BWLER in the linear PDE setting.

5.1 BWLER-hatted MLPs have smoother loss landscapes

Experimental setup. We start by evaluating BWLER-hatted MLPs on the three benchmark PDEs from Rathore et al. [31]: convection, reaction, and wave equations. Each model is trained using the Adam optimizer with identical network architecture and training settings. We compare three variants: a standard MLP, an (implicit) BWLER-hatted MLP, and an explicit BWLER model.

To probe convergence behavior beyond early training dynamics, we train each model for 10^6 iterations – significantly longer than prior work – to examine both the final precision after saturation and the consistency of convergence trends. Full experimental details are provided in Appendix C.2.

Results. Table 1 reports final ℓ_2 relative errors (L2RE) across all methods. BWLER-hatted MLPs consistently outperform standard MLPs, improving L2RE by around $30\times$ on the convection equation, $10\times$ on reaction, and $1800\times$ on wave. We replicate findings from prior work [22, 31] that pure MLPs often converge to suboptimal local minima when trained with Adam alone. For instance, on the convection equation (Section 1), the baseline MLP only recovers a single oscillation of the ground truth periodic solution; similarly, on the wave equation (Figure 6), the MLP recovers the high-level structure of the solution but not the fine-grained details. In contrast, the BWLER-hatted model finds a solution qualitatively matching the ground truth, and precision improves consistently with Adam alone (Figure 6).

Improved loss landscape conditioning. Towards understanding why BWLER improves optimization, we estimate the spectral density of the PINN loss's Hessian after convergence, following Rathore et al. [31]. We find that BWLER makes the loss landscape less ill-conditioned on the wave and reaction equations, reducing the maximum eigenvalue by $10\times$ and mean eigenvalue by 5–$10\times$ (Figures 8, 9). See Appendix C.2 for discussion and ablations.

Precision limitations of BWLER-hatted MLPs. Although BWLER-hatted MLPs outperform standard PINNs, their precision nonetheless plateaus more than 10 orders of magnitude L2RE worse than machine precision (Table 1) – mirroring the limitations seen in the interpolation setting (Section 3.2). We attribute this stagnation to the underlying MLP parameterization. In Section 5.2, we show that switching to an explicit representation and training with a preconditioned second-order method allows BWLER to overcome this barrier, achieving high precision solutions on the convection and wave equations.

L2RE $\downarrow$	MLP	BWLER-hatted MLP	BWLER
Convection	1.14×10^{0}	3.91×10^{-2} $(29.2\times)$	$\mathbf{4.07\times10^{-4}}$ $\mathbf{(2800\times)}$
Reaction	4.02×10^{-3}	$\mathbf{3.91\times10^{-4}}$ $\mathbf{(10.3\times)}$	7.10×10^{-2} $(0.057\times)$
Wave	5.22×10^{-1}	$\mathbf{2.88\times10^{-4}}$ $\mathbf{(1800\times)}$	9.99×10^{-1} $(0.52\times)$

Table 1: L2 relative errors (L2RE) on benchmark PDEs: convection, reaction, and wave equations from [31]. Multiplicative improvements (in parentheses) are relative to the MLP baseline. All models are trained with Adam for 10^6 iterations.

5.2 Explicit BWLER solves PDEs to high precision

Motivated by the precision limitations in the BWLER-hatted MLP setting, we next study explicit BWLER models. This formulation eliminates the extra precision bottlenecks introduced by the neural network parameterization – all ill-conditioning in the loss arises purely from the PDE and its discretization – allowing us to probe the precision limits of the physics-informed framework.

Optimization challenges. We begin by training explicit BWLER models with Adam alone on the three benchmark PDEs from Rathore et al. [31] (Table 1, rightmost column). While the models are expressive enough to precisely represent the true solution, they converge slowly under standard first-order optimizers. On the reaction equation, explicit BWLER underperforms even standard PINNs by a factor of $20\times$ in L2RE, and makes almost no progress during training on the wave equation. Note that although explicit BWLER with Adam outperforms standard PINNs by $2800\times$ and BWLER-hatted MLPs by $100\times$ on the convection equation, this is mostly due to the extremely high number of training steps (10^6 iterations with Adam). See Appendix C.2 for more detailed results.

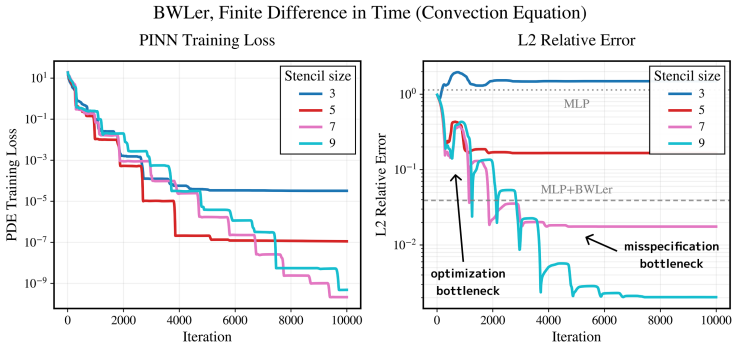

Figure 3: Precision-conditioning tradeoff. We train explicit BWLER models on the convection equation, using finite-difference derivatives in time, and vary the stencil size. Smaller stencils improve the problem's conditioning, improving initial convergence rate, but increase misspecification error, producing a precision saturation threshold.

Theory: convergence-conditioning tradeoff for 1-D linear differential operators. For explicit BWLER, the PDE setting admits an error decomposition mirroring the interpolation setting of Theorem 4.1. We present the 1-D linear setting, but note that the decomposition extends directly to higher-dimensional linear problems.

Theorem 5.1 (Precision-conditioning tradeoff for physics-informed BWLER, informal)**.** *We consider solving the d-th order PDE problem $Lu = 0$, where u satisfies the usual analyticity assumptions, by approximating L with a k-th order finite-difference scheme. Fitting an $(N+1)$-parameter* BWLER *via t steps of gradient descent on this discretized operator yields*

$$\|u - u_N^{(t)}\|_\infty \leq \underbrace{O(\rho^{-N})}_{\text{expressivity gap}} + \underbrace{\widetilde{O}\left(N^{-(k+1-d)}\right)}_{\text{bias/misspecification gap}} + \underbrace{\widetilde{O}\left(e^{-t/\kappa(N)^2}\right)}_{\text{optimization gap}}, \tag{5}$$

where $\rho > 1$ only depends on the function smoothness, and $\kappa(N)$ is the condition number of the discretized operator.

In Equation (5), the expressivity gap is the standard Cheybshev approximation bound (Theorem 2.2), the misspecification gap comes from the order of the finite-difference approximation [13], and the optimization gap is the standard gradient descent convergence rate on least squares [5]. Refer to Theorem D.8 and Appendix D for a formal theorem statement and proof.

Two precision-conditioning tradeoffs directly emerge from Theorem 5.1:

- **Expressivity vs. optimization.** The conditioning of order-d derivatives with spectral differentiation scales as $O(N^{2d})$ [32]. Decreasing the expressivity gap relies on increasing N, but this necessarily worsens the problem's conditioning and convergence rate.
- **Misspecification vs. optimization.** One way to improve the problem's conditioning is to try alternate derivative formulations, e.g. finite-difference schemes instead of spectral differentiation. Although FD matrices are better-conditioned (for 3-point stencils, $O(N^d)$ instead of $O(N^{2d})$ [13]), the misspecification gap increases in turn.

We note that a similar decomposition can be stated for nonlinear problems with standard PINNs, but a precise analysis of the precision-conditioning tradeoff is challenging [20, 28].

Training techniques towards efficient, high precision training. To navigate the precision-conditioning tradeoff, we combine the following three techniques:

- **Nyström-Newton-CG (NNCG)** [31]. NNCG is a second-order method that approximates the Newton step using a low-rank Nyström approximation to the Hessian. We tune the preconditioner rank and the number of CG iterations per Newton step to control the convergence rate.
- **Derivative quality tuning**. BWLER allows us to freely swap out different derivative computation methods. We experiment with spectral derivatives and finite differences where we vary the stencil size (*e.g.* 3-point stencil yields a 1st-order approximation, while a global stencil recovers the spectral derivative). See Appendix B for implementation details.
- **Multi-stage training.** Since BWLER is parameterized directly via its values on a discrete grid, we can warm-start training using any pretrained PINN (including another BWLER or a standard MLP).

High precision solutions to benchmark PDEs. Table 2 reports final ℓ_2 relative errors (L2RE) across five benchmark PDEs. On the convection, reaction, and wave equations, explicit BWLER achieves (near-)machine-precision solutions: 8–10 orders of magnitude improvements relative to the L2RE of PINNs reported in the literature. On Burgers' and an irregular-geometry Poisson problem, explicit BWLER matches the precision of prior state-of-the-art PINNs. Appendix C.2 provides implementation details, training diagnostics, and problem-specific strategies used to achieve these results.

We note that our results are *not* time- or parameter-matched: see Appendix C.2 for details. We view our results as a proof-of-concept: they show that machine-precision solutions are in fact possible within the PINN framework, but high precision requires careful codesign of models and optimizers, with the precision-conditioning tradeoff in mind. We also note that existing training techniques (*e.g.* NTK loss reweighting, temporal causality [37]) are complementary to our approach and likely could be leveraged to speed up training.

L2RE $\downarrow$	SOTA (from literature)	Explicit BWLER
Convection ($c=40$)	1.94×10^{-3} [31]	$\mathbf{2.04\times10^{-13}}$ $(9.51\times10^{9}\times)$
Convection ($c=80$)	6.88×10^{-4} [37]	$\mathbf{1.10\times10^{-12}}$ $(6.25\times10^{8}\times)$
Wave	1.27×10^{-2} [31]	$\mathbf{1.26\times10^{-11}}$ $(1.00\times10^{9}\times)$
Reaction	9.92×10^{-3} [31]	$\mathbf{6.94\times10^{-11}}$ $(1.43\times10^{8}\times)$
Burgers (1D-C)	1.33×10^{-2} [16]	$\mathbf{4.63\times10^{-3}}$ $(2.87\times)$
Poisson (2D-C)	1.23×10^{-2} [16]	$\mathbf{1.08\times10^{-2}}$ $(1.14\times)$

Table 2: L2 relative errors (L2RE) on benchmark PDEs problems. SOTA column reports (to our knowledge) the best results from the literature. Note: results are not time- or parameter-matched.

6 Conclusion

Discussion. Our results demonstrate that incorporating barycentric interpolants into the PINN framework dramatically improves attainable precision while maintaining the flexibility to handle diverse PDEs and complex geometries. On 1-D interpolation tasks, explicit BWLER models recover spectral convergence, reaching relative errors near 10^{-12}. BWLER-hatted MLPs, our drop-in variant, similarly boost precision by up to $10{,}000\times$ over standard MLPs (Figure 2). On PDE benchmarks, BWLER-hatted MLPs boost the precision of standard PINNs by up to $1800\times$ (Table 1). Using a second-order optimizer, we reach near-machine precision for convection, reaction, and wave equations (between 10^{-13}–10^{-11}), 8–10 orders of magnitude better than prior state-of-the-art. To our knowledge, this is the first instance of a PINN reaching machine-precision solutions even on 2-D problems.

Limitations. Despite these gains in precision, several limitations remain. First, the runtime cost of training is substantial. Because of their ill-conditioned loss landscapes, explicit BWLERs require

significantly longer to train than both traditional numerical solvers and prior PINN architectures. Although our results establish a new precision ceiling for PINNs, they do not yet outperform classic numerical methods in terms of precision per unit time. Second, BWLER thrives on PDEs with smooth solutions but performance deteriorates on stiff PDEs with sharp features or on irregular domains, settings where spectral methods traditionally struggle. Finally, our use of explicit grids may pose scalability challenges in higher-dimensional problems, where mesh-free methods often hold an advantage.

Acknowledgements

The authors thank Owen Dugan, Sabri Eyuboglu, Roberto Garcia, William Gilpin, Kade Heckel, Jeffrey Lai, Pratik Rathore, Benjamin Spector, Ben Viggiano, and Michael Zhang for their helpful feedback and discussion.

The authors gratefully acknowledge the support of NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF2247015 (Hardware-Aware), CCF1763315 (Beyond Sparsity), CCF1563078 (Volume to Velocity), and 1937301 (RTML); US DEVCOM ARL under Nos. W911NF-23-2-0184 (Long-context) and W911NF-21-2-0251 (Interactive Human-AI Teaming); ONR under Nos. N000142312633 (Deep Signal Processing); Stanford HAI under No. 247183; NXP, Xilinx, LETI-CEA, Intel, IBM, Microsoft, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, Google Cloud, Salesforce, Total, the HAI-GCP Cloud Credits for Research program, the Stanford Data Science Initiative (SDSI), and members of the Stanford DAWN project: Meta, Google, and VMWare. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of NIH, ONR, or the U.S. Government. JL is supported by the Department of Energy Computational Science Graduate Fellowship under Award Number DE-SC0023112. AR's research is supported by NSF grant CCF#2247014.

References

[1] Battles, Z. and Trefethen, L. N. An extension of matlab to continuous functions and operators. *SIAM Journal on Scientific Computing*, 25(5):1743–1770, 2004.

[2] Berman, J. and Peherstorfer, B. Randomized sparse neural galerkin schemes for solving evolution equations with deep networks. *Advances in Neural Information Processing Systems*, 36:4097–4114, 2023.

[3] Berrut, J.-P. and Trefethen, L. N. Barycentric lagrange interpolation. *SIAM Review*, 46(3): 501–517, 2004. doi: 10.1137/S0036144502417715. URL `https://doi.org/10.1137/S0036144502417715`.

[4] Boyd, J. P. *Chebyshev & Fourier Spectral Methods*. Lecture Notes in Engineering. Springer Berlin, Heidelberg, Berlin, Heidelberg, 1 edition, 1989. ISBN 978-3-540-51487-9.

[5] Boyd, S. P. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

[6] Brunton, S. L. and Kutz, J. N. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[7] Brunton, S. L., Nathan Kutz, J., Manohar, K., Aravkin, A. Y., Morgansen, K., Klemisch, J., Goebel, N., Buttrick, J., Poskin, J., Blom-Schieber, A. W., et al. Data-driven aerospace engineering: reframing the industry with machine learning. *Aiaa Journal*, 59(8):2820–2847, 2021.

[8] Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. *Spectral methods*, volume 285. Springer, 2006.

[9] Chen, H., Wu, R., Grinspun, E., Zheng, C., and Chen, P. Y. Implicit neural spatial representations for time-dependent pdes. In *International Conference on Machine Learning*, pp. 5162–5177. PMLR, 2023.

[10] Chen, Z., McCarran, J., Vizcaino, E., Soljačić, M., and Luo, D. Teng: Time-evolving natural gradient for solving pdes with deep neural nets toward machine precision. *arXiv preprint arXiv:2404.10771*, 2024.

[11] Cox, S. M. and Matthews, P. C. Exponential time differencing for stiff systems. *Journal of Computational Physics*, 176(2):430–455, 2002.

[12] Evans, L. C. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, Rhode Island, 2 edition, 2010. doi: 10.1090/gsm/019.

[13] Fornberg, B. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of computation*, 51(184):699–706, 1988.

[14] Fornberg, B. *A practical guide to pseudospectral methods*. Number 1. Cambridge university press, 1998.

[15] Frisch, U. *Turbulence: the legacy of AN Kolmogorov*. Cambridge university press, 1995.

[16] Hao, Z., Yao, J., Su, C., Su, H., Wang, Z., Lu, F., Xia, Z., Zhang, Y., Liu, S., Lu, L., and Zhu, J. Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes, 2023. URL https://arxiv.org/abs/2306.08827.

[17] Hughes, T. J. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2003.

[18] Karnakov, P., Litvinov, S., and Koumoutsakos, P. Optimizing a discrete loss (odil) to solve forward and inverse problems for partial differential equations using machine learning tools. *arXiv preprint arXiv:2205.04611*, 2022.

[19] Karnakov, P., Litvinov, S., and Koumoutsakos, P. Solving inverse problems in physics by optimizing a discrete loss: Fast and accurate learning without neural networks. *PNAS nexus*, 3(1): pgae005, 2024.

[20] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[21] Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. Characterizing possible failure modes in physics-informed neural networks. *Advances in neural information processing systems*, 34:26548–26560, 2021.

[23] Liu, D. C. and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[24] Liu, J. W., Grogan, J., Dugan, O. M., Rao, A., Arora, S., Rudra, A., and Re, C. Towards learning high-precision least squares algorithms with sequence models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.12295.

[25] McGovern, A., Elmore, K. L., Gagne, D. J., Haupt, S. E., Karstens, C. D., Lagerquist, R., Smith, T., and Williams, J. K. Using artificial intelligence to improve real-time decision-making for high-impact weather. *Bulletin of the American Meteorological Society*, 98(10):2073–2090, 2017.

[26] McGreivy, N. and Hakim, A. Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations. *Nature Machine Intelligence*, 6 (10):1256–1269, September 2024. ISSN 2522-5839. doi: 10.1038/s42256-024-00897-5. URL http://dx.doi.org/10.1038/s42256-024-00897-5.

[27] Michaud, E. J., Liu, Z., and Tegmark, M. Precision machine learning. *Entropy*, 25(1):175, January 2023. ISSN 1099-4300. doi: 10.3390/e25010175. URL http://dx.doi.org/10.3390/e25010175.

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

[29] Raissi, M., Perdikaris, P., and Karniadakis, G. E. Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683–693, November 2017. ISSN 0021-9991. doi: 10.1016/j.jcp.2017.07.050. URL http://dx.doi.org/10.1016/j.jcp.2017.07.050.

[30] Raissi, M., Perdikaris, P., and Karniadakis, G. 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. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[31] Rathore, P., Lei, W., Frangella, Z., Lu, L., and Udell, M. Challenges in training pinns: A loss landscape perspective, 2024. URL https://arxiv.org/abs/2402.01868.

[32] Trefethen, L. N. *Spectral methods in MATLAB*. SIAM, 2000.

[33] Trefethen, L. N. *Approximation Theory and Approximation Practice, Extended Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2019. doi: 10.1137/1.9781611975949. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611975949.

[34] Tropp, J. A. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics*, 12:389–434, 2012.

[35] Vershynin, R. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

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

[37] Wang, S., Sankaran, S., Wang, H., and Perdikaris, P. An expert's guide to training physics-informed neural networks, 2023. URL https://arxiv.org/abs/2308.08468.

[38] Wang, Y. and Lai, C.-Y. Multi-stage neural networks: Function approximator of machine precision, 2023. URL https://arxiv.org/abs/2307.08934.

[39] Wilcox, D. *Turbulence Modeling for CFD*. Number v. 1 in Turbulence Modeling for CFD. DCW Industries, 2006. ISBN 9781928729082. URL https://books.google.com/books?id=tFNNPgAACAAJ.

[40] Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*, pp. 581–590. IEEE, 2020.

A Related work

A.1 High-precision machine learning for PDEs

The difficulty of achieving high precision in machine learning for scientific applications is well-documented: despite progress from the scientific ML community in recent years, traditional numerical methods still outperform existing PDE learning approaches in precision, even on simple benchmark problems [26]. We are not aware of any physics-informed neural network approaches that obtain machine-precision solutions, even on standard 2-D linear PDE benchmarks.

Recent work in the PINN literature has explored architectural modifications, addressed optimization challenges, and proposed specialized training strategies [22, 37, 31]. The inherent precision limitations of existing ML architectures is comparatively underexplored. [27, 38] focus on the regression setting using MLPs and propose alternate training recipes towards higher precision. [24] identifies precision bottlenecks resulting from the Transformer architecture and standard LR schedulers in the setting of least squares.

Unlike prior work that focuses primarily on studying either PDE optimization or model architecture, in this paper we attempt to study both jointly:

- We demonstrate that the MLP parameterization of standard PINNs limits precision in the simple setting of 1-D function approximation, even without the additional challenges of PDE constraints (Section 3).
- We propose BWLER, which decouples model parameterization from derivative computation (Section 4). This allows us to separately study the precision limitations induced by the model versus the PDE conditioning.
- Using BWLER, we detail an explicit tradeoff between precision and conditioning in the linear PDE setting (Theorem 5.1). Along the way, we provide empirical evidence that our barycentric interpolants represent a simple yet surprisingly effective baseline parameterization for high-precision PDE learning; they achieve the high precision of traditional spectral methods on benchmark PDEs with smooth solutions, while maintaining compatibility with physics-informed frameworks (Tables 1, 2).

A.2 Hybrid approaches: combining PINNs with numerical methods

Recent work has explored integrating classical numerical techniques with machine learning-based PDE solvers to improve robustness, accuracy, and convergence. We highlight two relevant approaches:

- **Time-marching with PINNs.** Recent works attempt to embed numerical solvers directly into the training loop of PINNs, most commonly in handling time-dependent PDEs. For example, [22, 37] propose to divide the time domain into multiple subdomains and perform curriculum learning within a PINN framework to boost performance. Another set of approaches [9, 2, 10] directly incorporate time-stepping via classical integrators (e.g., Runge–Kutta) within a neural network framework to stabilize temporal dynamics, especially for stiff or chaotic systems.
- **ODIL.** The ODIL framework [18, 19] formulates PDE learning as the minimization of discretized residuals over mesh-based domains, preserving the structure and sparsity of finite volume and finite difference discretizations while enabling gradient-based optimization with neural networks.

  Like ODIL, our method also reintroduces an explicit grid under the hood of a physics-informed learning framework, but BWLER differs in two ways. **(1)** Just as ODIL exactly embeds finite volume and finite difference methods into an auto-differentiable ML setup, BWLER respectively embeds pseudo-spectral methods into physics-informed learning. This means that unlike the lower-precision PDE discretizations of finite differences, BWLER leverages the spectral convergence of polynomial approximation on smooth functions. **(2)** Additionally, BWLER can be flexibly treated both as a self-standing architecture or as an additional layer that goes atop existing PINN architectures.

A.3 Spectral methods and barycentric interpolation

Classical spectral methods, including Chebyshev and Fourier-based solvers, have long been used for high-accuracy PDE solutions on regular domains [4, 32]. These methods excel when the solution is smooth and the domain is simple, offering exponential convergence rates in both function and derivative approximation. Spectral element methods [8] extend these ideas to complex geometries by combining high-order accuracy with domain decomposition, and remain state-of-the-art in areas like fluid dynamics where both precision and geometric flexibility are critical [15, 39].

Recent frameworks like Chebfun [1] have revived interest in spectral approaches by enabling function-level computation with near-machine precision. Barycentric Lagrange interpolation [3] provides a numerically stable alternative to classical polynomial bases and serves as the foundation for many pseudo-spectral techniques. Our work is, to our knowledge, the first to integrate barycentric interpolation directly into a physics-informed learning framework.

Our approach closely parallels classical pseudo-spectral methods while introducing key flexibilities from the machine learning paradigm. Like traditional pseudo-spectral solvers, we represent the solution via its values at Chebyshev nodes, and we compute high-precision derivatives spectrally. However, BWLER additionally inherits the generality of the physics-informed paradigm:

- **Modern optimization and auto-differentiation.** BWLER seamlessly integrates with auto-differentiation frameworks on GPU-accelerated hardware. Instead of using classical iterative solvers, we can leverage ML optimizers such as Adam [21], L-BFGS [23], and NNCG [31].
- **Flexible derivative computation.** While classical solvers typically rely on differentiation matrices, BWLER allows switching between spectral (FFT-based) and finite difference derivatives to match the problem structure. See Appendix B for details.
- **Support for irregular geometries.** BWLER's barycentric formulation accommodates non-rectangular domains and complex boundary conditions using the least-squares framework of physics-informed learning. This avoids the manual domain transformations that traditional spectral methods need [4, 8].

B Method

In this section, we provide more details about the implementation of the BWLER architecture and training.

B.1 BWLER architecture

We provide full algorithmic details of the BWLER architecture. We first focus on the 1-D case before generalizing to higher dimensions. The core components are:

- **Chebyshev setting (1-D, non-periodic).** We describe barycentric interpolation and spectral differentiation using the Chebyshev-Gauss-Lobatto grid. This forms the foundation for interpolation and differentiation in non-periodic domains.
- **Fourier setting (1-D, periodic).** For periodic boundary conditions, we instead use Fourier nodes and basis functions. We describe both interpolation and differentiation with trigonometric polynomial interpolants.
- **Finite difference matrices.** As an alternative to spectral differentiation, we optionally use finite difference (FD) methods with Fornberg's algorithm to generate sparse banded derivative matrices.
- **Domain transformation.** All 1-D methods assume canonical domains ($[-1,1]$ for Chebyshev; $[0,2\pi]$ for Fourier), but are extended to arbitrary physical domains via affine coordinate maps.
- **Higher-dimensional extension.** We extend all components to multiple dimensions using tensor-product constructions, which factorize evaluation and differentiation along each axis.

This appendix provides explicit pseudocode for each of the above settings and highlights the computational properties relevant to their use in PINN frameworks.

B.1.1 Chebyshev setting, non-periodic

Evaluation. We begin with interpolation on the canonical Chebyshev-Gauss-Lobatto (CGL) grid. We consider interpolating a 1-D function $f:[-1,1]\to\mathbb{R}$. Let

$$x_j=\cos\left(\tfrac{j\pi}{N}\right),\quad w_j=(-1)^j\begin{cases}\frac{1}{2}, & j\in\{0,N\},\\ 1, & \text{otherwise},\end{cases}\qquad j=0,\ldots,N.$$

Then for any query $x\in[-1,1]$, we recall the barycentric formula (Equation (3)) gives

$$f_\theta(x)=\frac{\displaystyle\sum_{j=0}^{N}\frac{w_j f_j}{x-x_j}}{\displaystyle\sum_{j=0}^{N}\frac{w_j}{x-x_j}}.$$

Algorithm 2 represents a pseudocode description of the barycentric formula, which is how BWLER implements the *evaluation* operation required for the physics-informed framework.

Algorithm 2 BARYINTERP: Chebyshev barycentric interpolation

Input: $x \in [-1,1]$; node values $\{f_j\}_{j=0}^{N}$
Output: interpolated value $f_\theta(x)$

▷ Compute CGL nodes and weights (if not precomputed)
for $j = 0$ **to** N **do**
    $x_j \leftarrow \cos\left(\frac{j\pi}{N}\right)$
    $w_j \leftarrow (-1)^j \times \left(\frac{1}{2} \text{ if } j \in \{0,N\} \text{ else } 1\right)$

▷ Handle exact node case
for $j = 0$ **to** N **do**
    if $x = x_j$ **then**
        return f_j

▷ Accumulate barycentric sums

$$N_{\text{sum}} \leftarrow \sum_{j=0}^{N} \frac{w_j f_j}{x - x_j}, \quad D_{\text{sum}} \leftarrow \sum_{j=0}^{N} \frac{w_j}{x - x_j}$$

Return $f_\theta(x) = \dfrac{N_{\text{sum}}}{D_{\text{sum}}}$

Differentiation. We compute first-order derivatives at the Chebyshev nodes in $\mathcal{O}(N \log N)$ via the FFT. Algorithm 3 represents a pseudocode implementation of BWLER's *differentiation* operation.

Algorithm 3 CHEBFFTDERIVATIVE: spectral derivative at CGL nodes

Input: node values $\mathbf{u} = (u_0,...,u_N)$
Output: derivative $\mathbf{d} = \left(f'(x_0),...,f'(x_N)\right)$

▷ Mirror data (even extension)

$$V \leftarrow \left[u_0,u_1,...,u_N,u_{N-1},...,u_1\right]$$

▷ Forward FFT

$$\hat{V} \leftarrow \mathrm{FFT}(V)$$

▷ Differentiate in frequency space
for $k = 0$ **to** $2N-1$ **do**
    $k_{\text{eff}} \leftarrow \begin{cases} k, & k \le N, \\ k - 2N, & k > N, \end{cases}$
    $\hat{W}_k \leftarrow i k_{\text{eff}} \hat{V}_k$

▷ Inverse FFT

$$W \leftarrow \mathrm{IFFT}(\hat{W})$$

▷ Chain-rule correction
for $j = 1$ **to** $N-1$ **do**
    $d_j \leftarrow -W_j \,/\, \sqrt{1 - x_j^2}$
$d_0 \leftarrow \sum_{n=0}^{N} n^2 \hat{u}_n, \quad d_N \leftarrow \sum_{n=0}^{N} (-1)^{n+1} n^2 \hat{u}_n$
Return $\mathbf{d}$

B.1.2 Fourier setting, periodic

For periodic domains, we use N equispaced nodes

$$x_j = \frac{2\pi j}{N}, \quad j = 0,...,N-1,$$

instead of Chebyshev-Gauss-Lobatto nodes. We consider interpolating a 1-D function $f : [0,2\pi] \to \mathbb{R}$. To do so, we represent f by its discrete Fourier series.

Evaluation. For a trigonometric interpolant on equispaced nodes, *evaluation* can be performed using the FFT. We provide a pseudocode implementation in Algorithm 4.

Algorithm 4 FOURIERINTERP: trigonometric interpolation

Input: query point $x \in [0,2\pi]$; node values $\{f_j\}_{j=0}^{N-1}$
Output: interpolated value $f_\theta(x)$

▷ Precompute grid
for $j = 0$ **to** $N-1$ **do**
    $x_j \leftarrow \frac{2\pi j}{N}$

▷ Compute Fourier coefficients

$$\hat{f} \leftarrow \mathrm{FFT}\big(\{f_j\}\big)/N$$

▷ Evaluate interpolant

$$f_\theta(x) = \sum_{k=0}^{N-1} \hat{f}_k e^{ikx}$$

Return $f_\theta(x)$

Differentiation. To perform differentiation on equispaced nodes, we define an explicit differentiation matrix, following [32]. Specifically, the Fourier differentiation matrix on N equispaced points $x_j = 2\pi j/N$ is defined as:

$$(D_N)_{ij} = \begin{cases} 0, & i = j, \\ \dfrac{(-1)^{i-j}}{2} \cot\Big(\frac{\pi(i-j)}{N}\Big), & i \neq j. \end{cases} \tag{6}$$

We provide a pseudocode implementation of *differentiation* for trigonometric interpolants in Algorithm 5.

Algorithm 5 FOURIERDERIVATIVEMATRIX: periodic derivative via explicit matrix

Input: node values $\mathbf{u} = (u_0,...,u_{N-1})$
    ▷ on equispaced grid $x_j = 2\pi j/N$
Output: derivatives $\mathbf{d} = (f'(x_0),...,f'(x_{N-1}))$

▷ Assemble Fourier differentiation matrix (Equation (6))
for $i = 0$ **to** $N-1$ **do**
    for $j = 0$ **to** $N-1$ **do**
        if $i = j$ **then**
            $(D)_{ij} \leftarrow 0$
        else
            $(D)_{ij} \leftarrow \frac{(-1)^{i-j}}{2} \cot\big(\frac{\pi(i-j)}{N}\big)$

▷ Apply matrix to values

$$\mathbf{d} \leftarrow D\mathbf{u}$$

Return $\mathbf{d}$

B.1.3 Finite–difference differentiation matrices

For arbitrary node distributions $\{x_j\}_{j=0}^N$, we employ Fornberg's algorithm [13] to construct a sparse, banded matrix $D^{(m,k)} \in \mathbb{R}^{(N+1)\times(N+1)}$ that approximates the m-th derivative using a stencil of half-bandwidth k. The entries satisfy

$$(D^{(m,k)}\mathbf{u})_i = \sum_{j=\max(0,i-k)}^{\min(N,i+k)} w_{ij}^{(m)} u_j,$$

and yield $O(h^{2k-m})$ accuracy on non-uniform grids. In the periodic setting, we recover the standard finite difference stencils on equispaced nodes.

The algorithm for performing differentiation using finite difference instead of spectral derivatives is outlined in pseudocode in Algorithm 6.

Algorithm 6 FDDERIVATIVEMATRIX: Fornberg finite-difference derivative

Input: node locations $\{x_j\}_{j=0}^N$, values $\mathbf{u}\in\mathbb{R}^{N+1}$, derivative order m, half-bandwidth k
Output: $\mathbf{d} = (f^{(m)}(x_0),\ldots,f^{(m)}(x_N))$

▷ Build differentiation matrix via Fornberg's method
$D^{(m,k)} \leftarrow \mathrm{FornbergMatrix}(\{x_j\},m,k)$ [13]

▷ Apply to node values
$\mathbf{d} \leftarrow D^{(m,k)}\mathbf{u}$

Return $\mathbf{d}$

B.1.4 Domain transformation to arbitrary intervals

All of the above 1-D formulas assume canonical domains ($[-1,1]$ for Chebyshev, $[0,2\pi]$ for Fourier). To handle a physical interval $[a,b]$, we apply an affine map $x\mapsto\tilde{x}$:

$$\tilde{x} = \begin{cases} \dfrac{2(x-a)}{b-a} - 1, & \text{Chebyshev,} \\ \dfrac{2\pi(x-a)}{b-a}, & \text{Fourier.} \end{cases}$$

All node locations, weights, and differentiation matrices are computed in $\tilde{x}$-space, and final function values or derivatives are re-mapped to the physical coordinate x. This preserves both the interpolation accuracy and spectral convergence properties on arbitrary intervals. We note that we must account for the rescaling factor when mapping function values to and from the physical and canonical domains.

B.1.5 Extension to higher dimensions

Let $\mathbf{x} = (x_1,\ldots,x_d)\in\Omega\subset\mathbb{R}^d$. We construct a tensor-product interpolant:

$$f(\mathbf{x}) = \sum_{j_1=0}^{N_1}\cdots\sum_{j_d=0}^{N_d} f_{j_1,\ldots,j_d}\prod_{\ell=1}^{d}\phi_{j_\ell}^{(\ell)}(x_\ell),$$

where each $\phi^{(\ell)}$ is the 1-D Chebyshev or Fourier barycentric basis on the ℓ-th axis. Evaluation and differentiation factorize along each dimension:

$$\partial_{x_1}^{k_1}\cdots\partial_{x_d}^{k_d} f(\mathbf{x}) = \sum_{j_1,\ldots,j_d} f_{j_1,\ldots,j_d}\prod_{\ell=1}^{d}\big(\phi_{j_\ell}^{(\ell)}\big)^{(k_\ell)}(x_\ell).$$

Thus, in practice, BWLER applies the 1-D interpolation or derivative operators sequentially (or, for differentiation matrices, via Kronecker-product routines) to achieve efficient interpolation and differentiation in higher dimensions.

B.2 Training

Our training algorithm consists of two key components: the optimizer for updating model parameters and the scheme for selecting collocation points where PDE constraints are enforced.

Optimizer. We experiment with two different optimizers:

- **Adam** [21]: The standard first-order optimizer in deep learning. By default, we use an initial learning rate of 10^{-3} with cosine decay learning rate schedule with a minimum learning rate of 10^{-6}.
- **Nyström-Newton CG** [31]: A specialized second-order method designed for PINNs that approximates the Hessian using Nyström sampling. We use the default hyperparameters from Rathore et al. [31] except for the rank of the preconditioner and the number of CG steps per Newton update, which we tune per problem. See Appendix C.2 for problem-specific hyperparameters.

Collocation scheme. For selecting collocation points where the PDE residual is enforced, we explore two strategies:

- **Random sampling.** Following standard PINN practice, we sample collocation points at each iteration. We compare two distributions:
  - Uniform sampling on $[-1,1]$: $x \sim \text{Unif}([-1,1])$
  - Chebyshev-weighted sampling: $x = \cos(\theta)$ where $\theta \sim \text{Unif}([0,\pi])$, which has density $\propto 1/\sqrt{1-x^2}$

  We default to the latter as it matches the node distribution in the model parameterization.
- **Fixed nodal collocation.** Unlike traditional PINNs which require dense sampling to ensure the PDE holds everywhere, our polynomial representation allows us to enforce the PDE only at the Chebyshev nodes $\{x_j\}_{j=0}^{N}$.

We find that nodal collocation suffices for the benchmark PDE problems we consider in this work. See Appendix C.2 for more details about hyperparameters for specific experiments.

L2 Relative Error Formula. For assessing the quality of interpolants and PDE solutions of all models used in this paper we leverage the standard ℓ_2 relative error (L2RE):

$$\text{L2RE}(f_\theta, f) = \frac{\|f_\theta - f\|_2}{\|f\|_2} = \sqrt{\frac{\sum_{i=1}^{N_{\text{test}}} (f_\theta(x_i) - f(x_i))^2}{\sum_{i=1}^{N_{\text{test}}} f(x_i)^2}}. \tag{7}$$

C Additional experimental details

C.1 1-D Interpolation

Here, we describe the experimental setup for our 1-D interpolation experiments (Section 3).

C.1.1 Task description

To study interpolation across functions of varying smoothness, we consider sinusoids $f(x) = \sin(kx)$, $x \in [-1,1]$, with varying frequency k. These serve as a controlled test case for examining how model precision scales with oscillatory complexity. We vary $k \in \{1,2,4,8,16,32\}$. For each target function, we generate a training set of $N_{train} = 100$ points sampled uniformly at random from the domain, and evaluate on a dense equispaced test grid of $N_{test} = 1000$ points. We run our interpolation experiments over five seeds, and report the median three results.

C.1.2 Model architecture and optimizer details

We compare standard MLPs, BWLER-hatted MLPs, and explicit BWLERs on the 1-D interpolation task. We train all models to minimize MSE using the Adam optimizer and use a cosine decay learning rate scheduler with a minimum learning rate of 10^{-6}.

Standard MLPs We use fully-connected MLPs with `tanh` activations. We sweep network widths within $\{2^4,...,2^8\}$ and depths from $2-8$ layers. We choose our initial learning rate by sweeping LR for the smallest MLP, and adjust the LR for larger MLPs by decreasing the initial learning rate by $\sqrt{ab}$ whenever we scale up the width by a factor of $a\times$ and the depth by a factor of $b\times$. Our base LR for the smallest MLP is 0.05.

BWLER-hatted MLPs We apply our BWLER-hats atop standard fully-connected MLPs with `tanh` activations, with width 256 and 3 hidden layers. We evaluate how precision scales with N, the number of nodes in the BWLER-hat, as we vary $N \in \{2^0,...,2^6\}$. We use an initial LR of 0.05 for our BWLER-hatted MLPs.

Explicit BWLERs As with BWLER-hatted MLPs, we sweep $N \in \{2^0,...,2^6\}$ and evaluate the precision scaling. We use an initial LR of 0.01 for all our explicit BWLERs.

C.1.3 Chebyshev least squares

As a classical baseline for function interpolation, we fit Chebyshev polynomials via least squares regression. Given a target function f, we construct a design matrix $A \in \mathbb{R}^{N_{\text{train}} \times (d+1)}$, where each row contains the values of the first $d+1$ Chebyshev polynomials $T_0(x),...,T_d(x)$ evaluated at a training point x_i. We then solve the linear system $Ac \approx f$ in the least-squares sense, where $c \in \mathbb{R}^{d+1}$ are the polynomial coefficients. Note that this baseline performs polynomial interpolation in *coefficient space*, whereas explicit BWLER performs polynomial interpolation in *value space* [33].

We implement this using NumPy's `numpy.polynomial.chebyshev.chebfit` function to fit the coefficients on the training data, and `chebval` for evaluation on the test grid. This provides an efficient and numerically stable method for approximating smooth functions, and serves as a reference for assessing model convergence in Section 3. Interestingly, we find that as the least squares problem becomes more ill-conditioned (i.e. as the degree of the polynomial N approaches the dataset size M), our explicit BWLER sometimes outperforms the least squares baseline on the test data (Figure 4). We attribute this to the early-stopping regularization effect of gradient descent on ill-conditioned least squares [5].

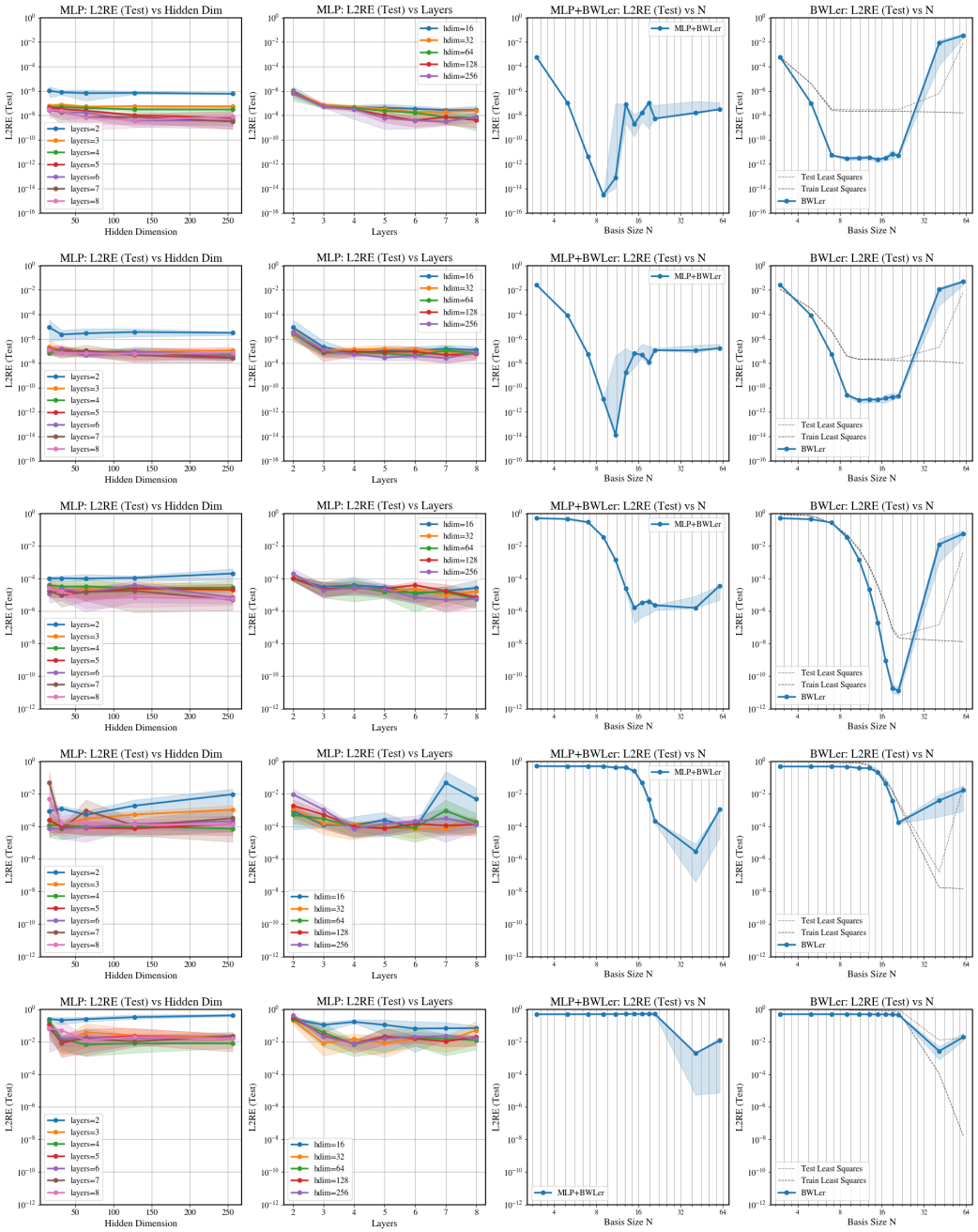

Figure 4: Comparison of standard MLPs, BWLER-hatted MLPs, and explicit BWLERs on 1-D interpolation with the target functions $f(x) = \sin(kx)$. From top to bottom: $k = 1, 2, 4, 16, 32$. Chebyshev least squares baseline plotted in dotted line on rightmost plots.

C.2 PDEs

C.2.1 Benchmark problems

We perform our experiments on five benchmark PDE problems from prior work:

Convection Equation. The one-dimensional convection equation is a first-order hyperbolic PDE commonly used to model phenomena in fluids, physics, and biology. We use the problem formulation

from Rathore et al. [31] and Wang et al. [37]:

$$\begin{aligned}\frac{\partial u}{\partial t}+c\frac{\partial u}{\partial x}&=0, \quad x\in(0,2\pi), t\in(0,1),\\ u(x,0)&=\sin(x), \quad x\in[0,2\pi],\\ u(0,t)&=u(2\pi,t), \quad t\in[0,1].\end{aligned}$$

The analytical solution is $u(x,t)=\sin(x-ct)$, where we set $c=40,80$ in our experiments.

Reaction Equation. The one-dimensional reaction equation is a non-linear ODE that models chemical reactions. We use the problem formulation from Rathore et al. [31]:

$$\begin{aligned}\frac{\partial u}{\partial t}-\rho u(1-u)&=0, \quad x\in(0,2\pi), t\in(0,1),\\ u(x,0)&=\exp\left(-\frac{(x-\pi)^2}{2(\pi/4)^2}\right), \quad x\in[0,2\pi],\\ u(0,t)&=u(2\pi,t), \quad t\in[0,1].\end{aligned}$$

The analytical solution is $u(x,t)=\frac{h(x)e^{\rho t}}{h(x)e^{\rho t}+1-h(x)}$, where $h(x)=\exp\left(-\frac{(x-\pi)^2}{2(\pi/4)^2}\right)$ and $\rho=5$ in our experiments.

Wave Equation. The one-dimensional wave equation is a second-order hyperbolic PDE that models wave propagation. We use the problem formulation from Rathore et al. [31]:

$$\begin{aligned}\frac{\partial^2 u}{\partial t^2}-4\frac{\partial^2 u}{\partial x^2}&=0, \quad x\in(0,1), t\in(0,1),\\ u(x,0)&=\sin(\pi x)+\frac{1}{2}\sin(\beta\pi x), \quad x\in[0,1],\\ \frac{\partial u(x,0)}{\partial t}&=0, \quad x\in[0,1],\\ u(0,t)&=u(1,t)=0, \quad t\in[0,1].\end{aligned}$$

The analytical solution is $u(x,t)=\sin(\pi x)\cos(2\pi t)+\frac{1}{2}\sin(\beta\pi x)\cos(2\beta\pi t)$, where $\beta=5$ in our experiments.

Burgers' Equation. The one-dimensional viscous Burgers' equation is a nonlinear PDE often used as a prototype for modeling shock waves. We follow the problem formulation from Hao et al. [16]:

$$\begin{aligned}\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}&=\nu\frac{\partial^2 u}{\partial x^2}, \quad x\in(-1,1), t\in(0,1),\\ u(x,0)&=-\sin(\pi x), \quad x\in[-1,1],\\ u(-1,t)&=u(1,t)=0, \quad t\in[0,1].\end{aligned}$$

We use $\nu=\frac{0.01}{\pi}$ in our experiments.

Poisson Equation. We consider the Poisson equation

$$-\Delta u=0,$$

on an irregular domain with four circular holes, following the setup in Hao et al. [16]. The domain is defined as a square with four circular cutouts:

$$\Omega=\Omega_{\text{rec}}\setminus\bigcup_i R_i, \quad \text{where } \Omega_{\text{rec}}=[-0.5,0.5]^2,$$

and the four circles are:

$$\begin{aligned}R_1&=\{(x,y):(x-0.3)^2+(y-0.3)^2\leq 0.1^2\},\\ R_2&=\{(x,y):(x+0.3)^2+(y-0.3)^2\leq 0.1^2\},\\ R_3&=\{(x,y):(x-0.3)^2+(y+0.3)^2\leq 0.1^2\},\\ R_4&=\{(x,y):(x+0.3)^2+(y+0.3)^2\leq 0.1^2\}.\end{aligned}$$

The boundary conditions are:

$$u = 0, \quad x \in \partial R_i,$$
$$u = 1, \quad x \in \partial \Omega_{\text{rec}}.$$

C.2.2 Results with BWLER-hatted MLPs

Experiment setup.

- *Benchmark PDE problems.* We compare standard MLPs vs. BWLER-hatted MLPs vs. explicit BWLERs on the convection, reaction, and wave equation benchmarks from Rathore et al. [31]. Details are described in Appendix C.2.1.
- *Model settings.* All the MLPs we use for the standard and BWLER-hatted MLP experiments use 3 layers and a hidden dimension of 256. The BWLER-hatted MLPs and explicit BWLER models use the problem-specific BWLER hyperparameters described in Appendix C.2.3.
- *Optimization settings.* We train all models using Adam [21] for 10^6 iterations. We use an initial learning rate of 10^{-3} and a cosine annealing learning rate schedule with a minimum learning rate of 10^{-6}. We use the standard momentum hyperparameters $(\beta_1, \beta_2) = (0.9, 0.999)$.

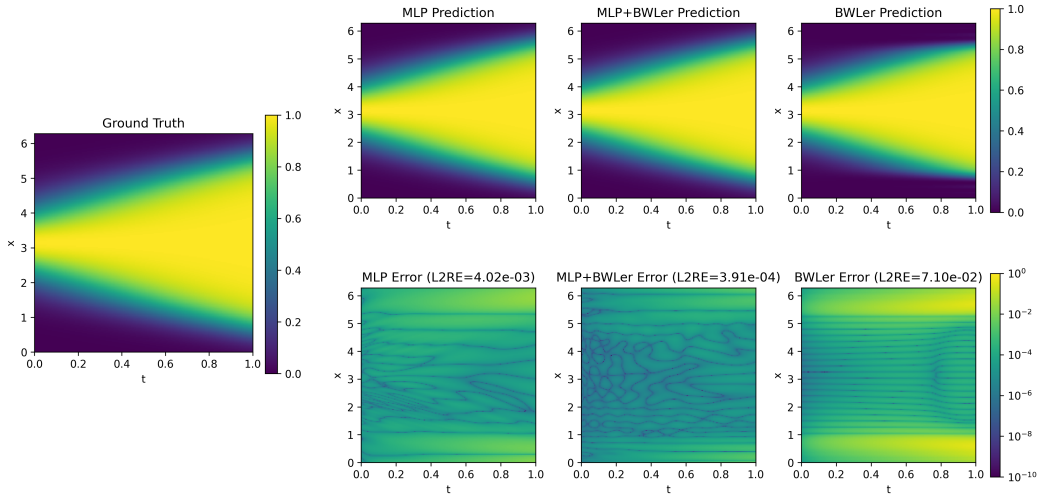

Figure 5: Standard MLP vs. BWLER-hatted MLP vs. explicit BWLER, evaluated on the reaction equation. For all models, we train for 10^6 iterations with Adam.

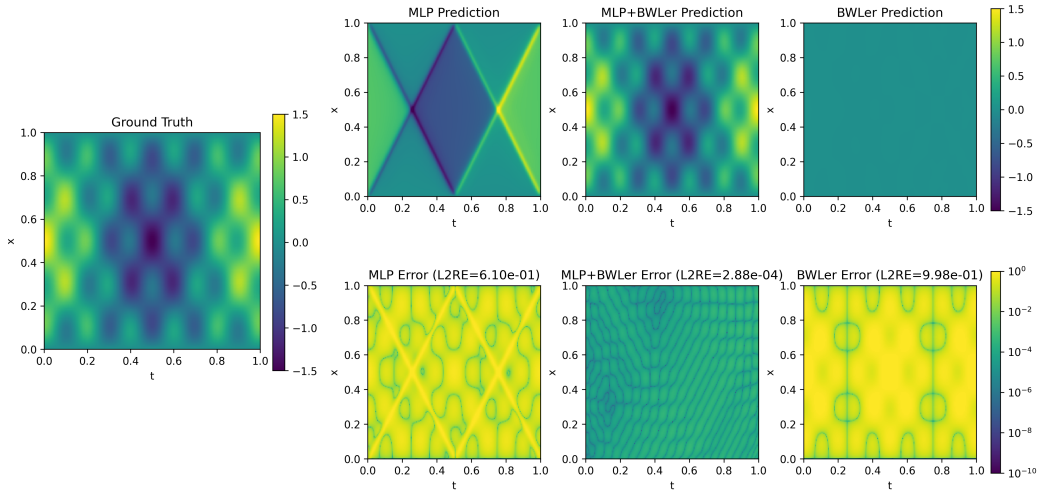

Figure 6: Standard MLP vs. BWLER-hatted MLP vs. explicit BWLER, evaluated on the wave equation. For all models, we train for 10^6 iterations with Adam.

BWLER inherits the shortcomings of spectral methods. Although BWLER can be flexibly applied to problems with complex boundary conditions and irregular domains, like standard PINNs, we do not expect BWLER-hatting to provide a consistent boost in performance across all PDE problems. Since BWLER's performance guarantees depend on the smoothness of the target function, like standard polynomial approximation methods, it exhibits similar shortcomings to spectral solvers.

To highlight this, we compare the performance of a standard MLP vs. a BWLER-hatted MLP and explicit BWLER on Burgers' equation, commonly used as a toy problem for shock capturing. This is an adversarial test problem for spectral methods, as the solution is nearly discontinuous; standard results about polynomial approximation imply approximation error should converge as $O(1/N)$ [33], where N is the number of nodes used in BWLER. We note that our explicit BWLER is equivalent to treating the 1+1D Burgers' equation spectrally in both space *and time*. This is unorthodox and suboptimal; a more standard approach is treating space spectrally and performing time marching, e.g. via Exponential Time Differencing [11].

We provide the results in Table 3, alongside the results for the convection, reaction, and wave equations from Table 1 for comparison. We train with 10^6 iterations of Adam for all methods. We indeed find that BWLER's global treatment of the solution *boosts* performance for smooth solutions, like the convection, reaction, and wave equations, but *worsens* performance for the nearly-discontinuous solution of Burgers' equation.

L2RE ↓	MLP	BWLER-hatted MLP	BWLER
Convection	1.14×10^{0}	3.91×10^{-2} (29.2×)	$\mathbf{4.07\times10^{-4}}$ **(2800×)**
Reaction	4.02×10^{-3}	$\mathbf{3.91\times10^{-4}}$ **(10.3×)**	7.10×10^{-2} (0.057×)
Wave	5.22×10^{-1}	$\mathbf{2.88\times10^{-4}}$ **(1800×)**	9.99×10^{-1} (0.52×)
Burgers'	$\mathbf{4.99\times10^{-3}}$	2.43×10^{-1} (0.021×)	9.49×10^{-1} (0.005×)

Table 3: L2 relative errors (L2RE) on benchmark PDEs: convection, reaction, and wave equations from Rathore et al. [31], and Burgers' equation from Hao et al. [16]. Multiplicative improvements (in parentheses) are relative to the MLP baseline, where a factor less than 1 means a *worse* performance than the standard MLP. All models are trained with Adam for 10^6 iterations.

Ablation: effect of BWLER-hatted MLP evaluation and differentiation. Note that when applying BWLER-hatting to an MLP, we can independently choose to use either BWLER's or the standard MLP's *evaluate* and *differentiate* operations. By default, our BWLER-hatted MLPs use BWLER for both evaluation and differentiation when solving PDEs. Here, we ablate the effect of two BWLER-hatted MLP variants:

- BWLER-*hatted MLP, forward only.* Uses BWLER's interpolation for evaluation but auto-differentiation of the MLP parameterization for the PDE derivatives.
- BWLER-hatted MLP, derivative only. Uses the standard MLP forward pass for evaluation but spectral derivatives from BWLER for the PDE derivatives.

We compare to the standard BWLER-hatted MLP, which uses BWLER for both evaluation and differentiation.

We evaluate on the convection equation from Rathore et al. [31], where the standard MLP only recovers a single oscillation of the true PDE solution, but the standard BWLER-hatted MLP recovers a qualitatively correct global solution (Table 1). Interestingly, we find both variants of BWLER-hatting, forward only and derivative only, *fail* to recover the global solution that the standard BWLER-hatted MLP does. This result supports our hypothesis that BWLER-hatting improves the ill-conditioning of the loss landscape by enforcing global consistency. The locality of the MLP, even when used only for evaluation alone or for differentiation alone, appears to disturb this effect, causing the ablation variants to converge to suboptimal local minima just like the standard MLP.

L2RE ↓	MLP	BWLER-hatted MLP (full)	BWLER-hatted MLP (forward only)	BWLER-hatted MLP (deriv only)
Convection	1.14×10^{0}	$\mathbf{3.91\times10^{-2}}$ **(29.2×)**	9.59×10^{-1} (1.19×)	9.59×10^{-1} (1.19×)

Table 4: L2 relative errors (L2RE) of standard MLP and BWLER-hatted MLP variants on convection PDE from Rathore et al. [31]. Multiplicative improvements (in parentheses) are relative to the MLP baseline. All models are trained with Adam for 10^6 iterations.

Hessian spectral density plots. Here, we include plots of the Hessian spectral density as described in Section 5.1. We compare standard MLPs, BWLER-hatted MLPs, and explicit BWLERs on the convection, reaction, and wave equations, trained for 10^6 iterations with Adam (Table 1). After training, we take the final trained models and approximate the Hessian spectral density for each using PyHessian [40]; it implements the stochastic Lanczos algorithm and uses Hessian-vector products.

We find that BWLER-hatting reduces the maximum eigenvalue by $10\times$ and the mean eigenvalue by 5–$10\times$ on the reaction and wave equations (Figure 8, Figure 9). This supports our hypothesis that BWLER's *evaluate* and *differentiate* operations, which depend globally on function values across the full domain, induce a less ill-conditioned loss landscape.

Interestingly, we find that BWLER-hatting worsens the conditioning on the convection equation compared to the standard MLP (Figure 7) – but this is because the standard MLP converges to a suboptimal local minima which is surprisingly effective at minimizing the PINN loss. See Figures 10 and 1 for visualizations.

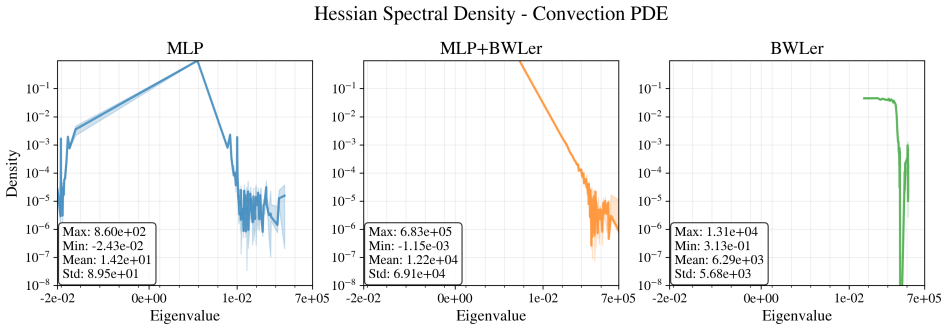

Figure 7: Hessian spectral density for the convection equation.

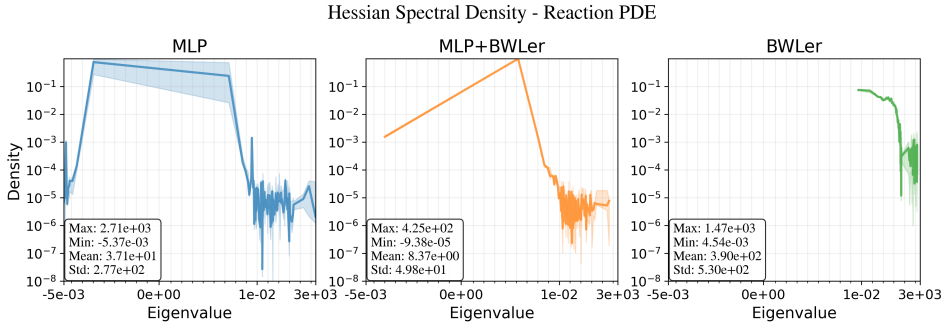

Figure 8: Hessian spectral density for the reaction equation.

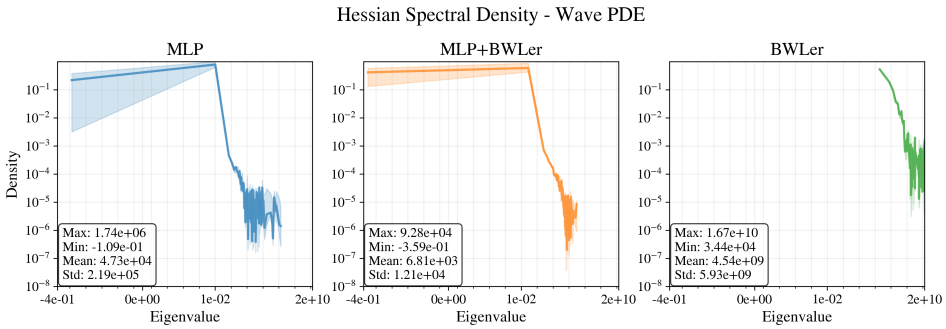

Figure 9: Hessian spectral density for the wave equation.

Loss curves. We provide loss curves for the experiments comparing standard MLPs, BWLER-hatted MLPs, and explicit BWLERs trained with Adam (Table 1).

Convection Equation.

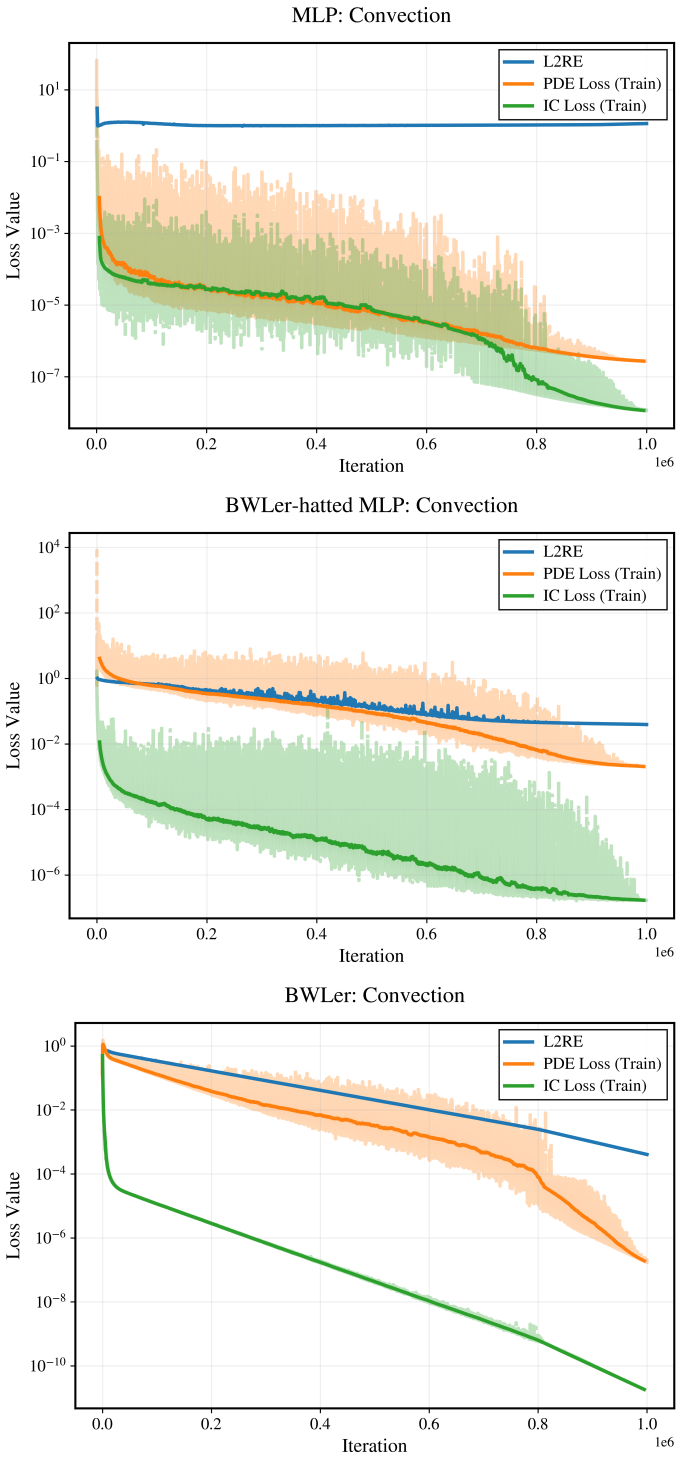

Figure 10: Loss curves for standard MLP, BWLER-hatted MLP, and explicit BWLER trained with Adam on convection equation with $c = 40$ (Table 1).

Reaction Equation.

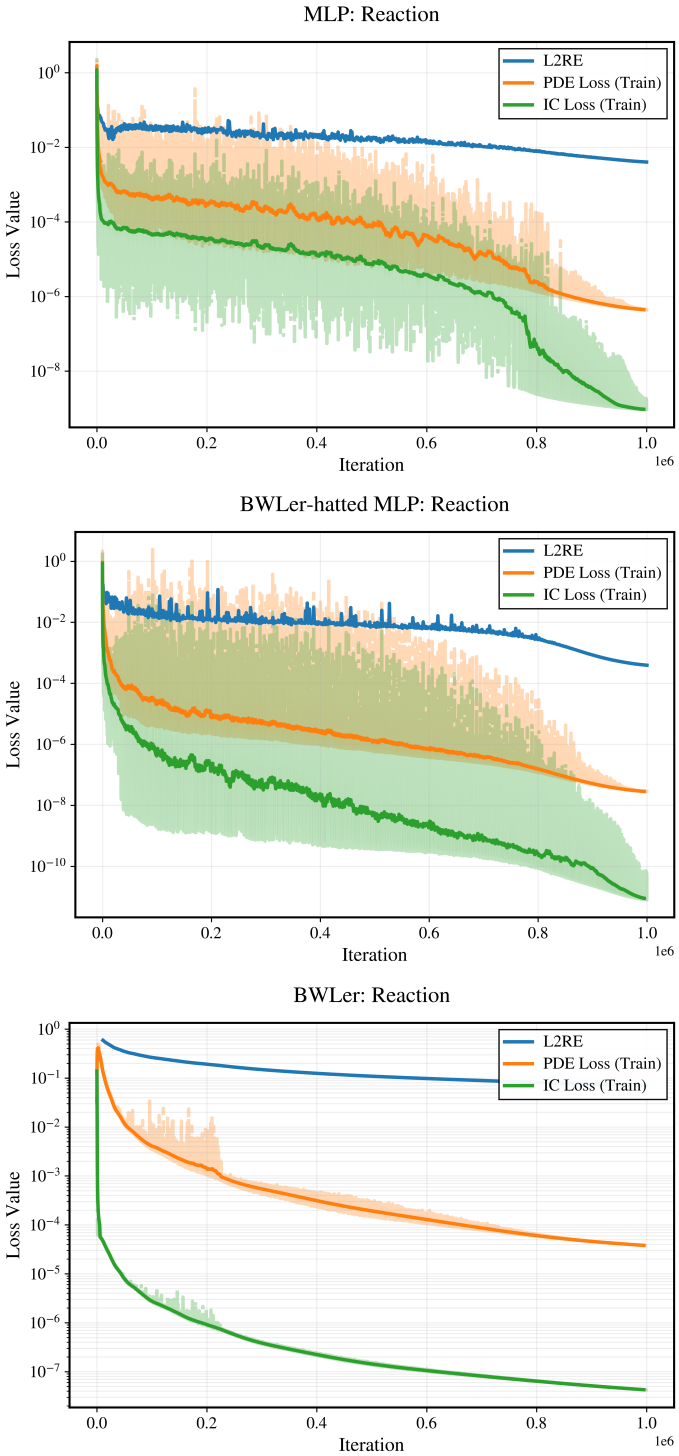

Figure 11: Loss curves for standard MLP, BWLER-hatted MLP, and explicit BWLER trained with Adam on reaction equation (Table 1).

Wave Equation.

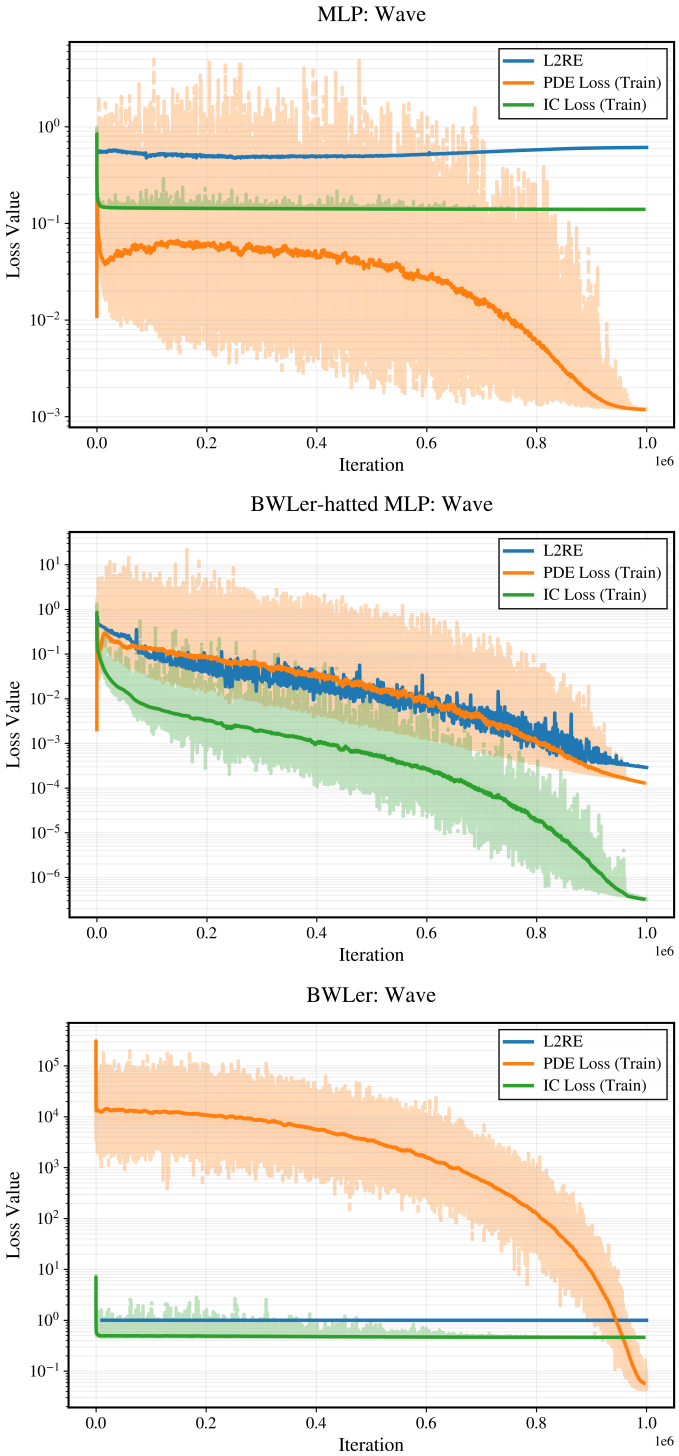

Figure 12: Loss curves for standard MLP, BWLER-hatted MLP, and explicit BWLER trained with Adam on wave equation (Table 1).

C.2.3 Results with explicit BWLERs

We describe the problem-specific BWLER architecture hyperparameter settings used in Tables 1, 2, and the high-precision optimization settings for the five benchmark PDE problems in Table 2. For each problem, we provide the loss curves, final learned solutions, and error residuals of the explicit BWLER experiments from Table 2.

Convection Equation, $c = 40$**.**

- *Architecture.* We use $N_t = 81$, $N_x = 80$, where we treat time with a Chebyshev basis and space with a Fourier basis.
- *High-precision optimization.* We train with Nyström-Newton-CG for 350 steps, with a preconditioner rank of 1000 and 100 CG iterations per step. On an A100, the total training takes about 5 minutes (about 1.2 iterations per second).

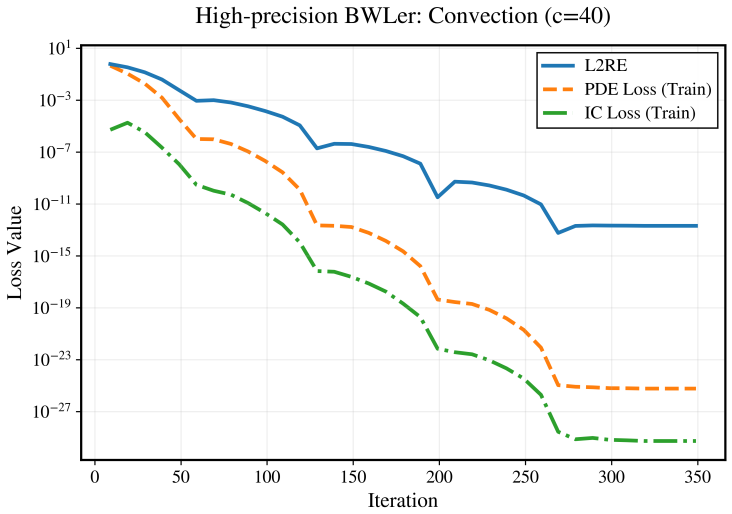

Figure 13: Loss curves for explicit BWLER trained with NNCG on convection equation with $c = 40$ (Table 2).

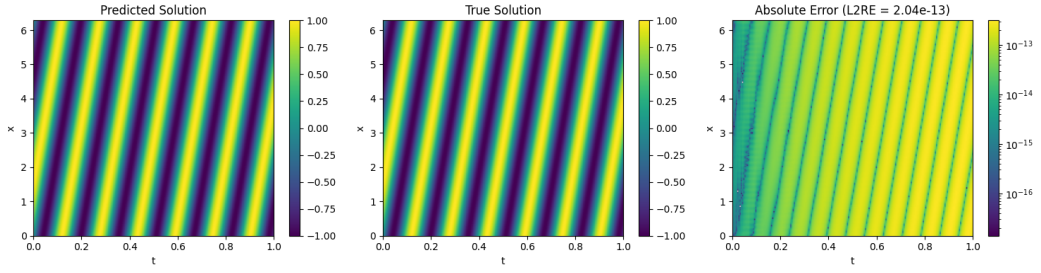

Figure 14: Explicit BWLER's learned solution and error residual on convection equation with $c = 40$ (Table 2).

Convection Equation, $c = 80$.

- *Architecture.* We use $N_t = 161$, $N_x = 160$, where we treat time with a Chebyshev basis and space with a Fourier basis.
- *High-precision optimization.* We train with Nyström-Newton-CG for 2500 steps, with a preconditioner rank of 1000 and 100 CG iterations per step. On an A100, the total training takes about 30 minutes (about 1.4 iterations per second).

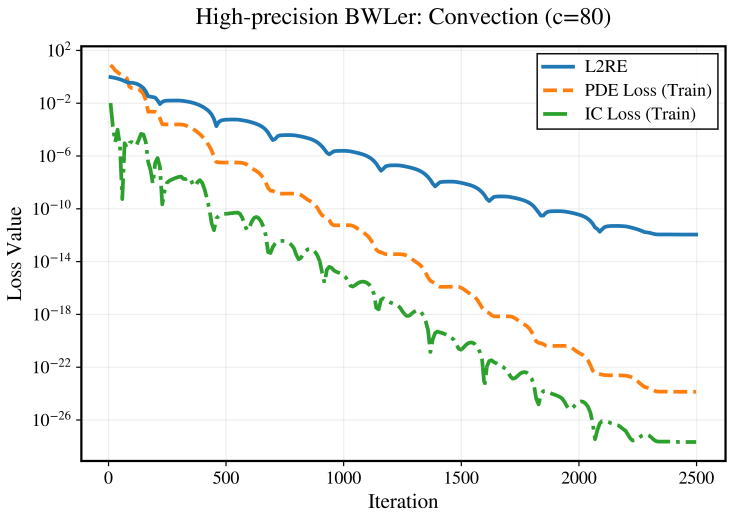

Figure 15: Loss curves for explicit BWLER trained with NNCG on convection equation with $c = 80$ (Table 2).

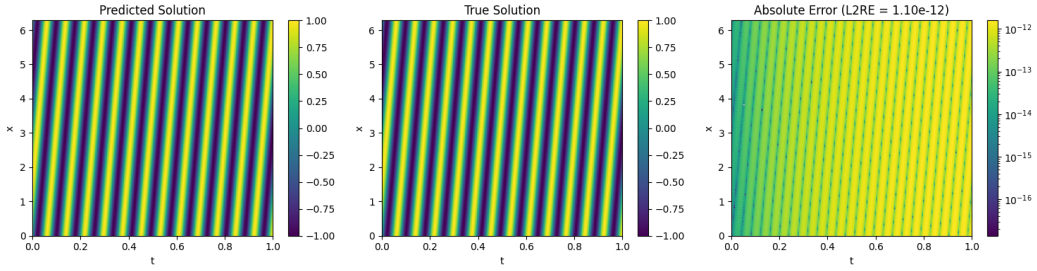

Figure 16: Explicit BWLER's learned solution and error residual on convection equation with $c = 80$ (Table 2).

Reaction Equation.

- *Architecture.* We use $N_t = N_x = 81$, where we treat both time and space with a Chebyshev basis.
- *High-precision optimization.* We train with Nyström-Newton-CG for $250{,}000$ steps, with a preconditioner rank of 16 and 16 CG iterations per step. On an A100, the total training takes about 8.5 hours (about 8.2 iterations per second).

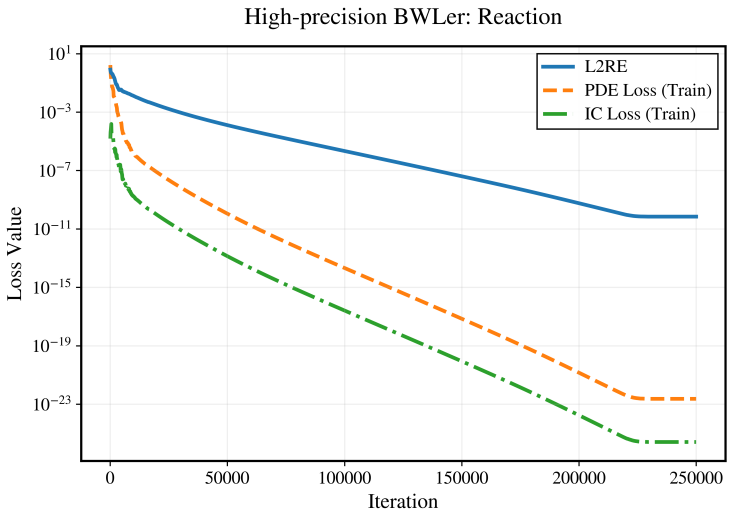

Figure 17: Loss curves for explicit BWLER trained with NNCG on reaction equation (Table 2).

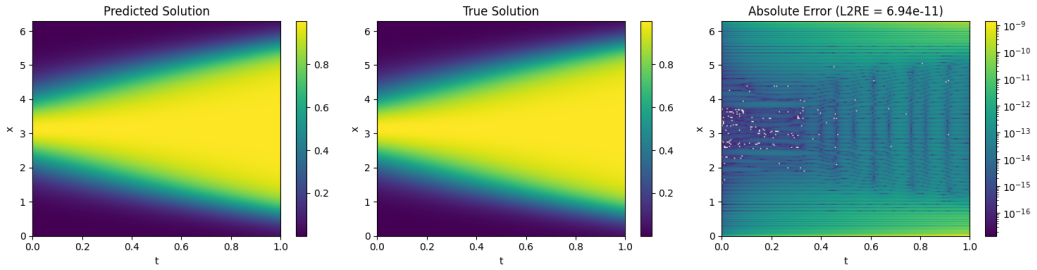

Figure 18: Explicit BWLER's learned solution and error residual on reaction equation (Table 2).

Wave Equation.

- *Architecture.* We use $N_t = N_x = 41$, where we treat both time and space with a Chebyshev basis.
- *High-precision optimization.* We train with Nyström-Newton-CG for 200 steps, with a preconditioner rank of 1000 and 1000 CG iterations per step. On an A100, the total training takes about 42 minutes (about 12.5 seconds per iteration).

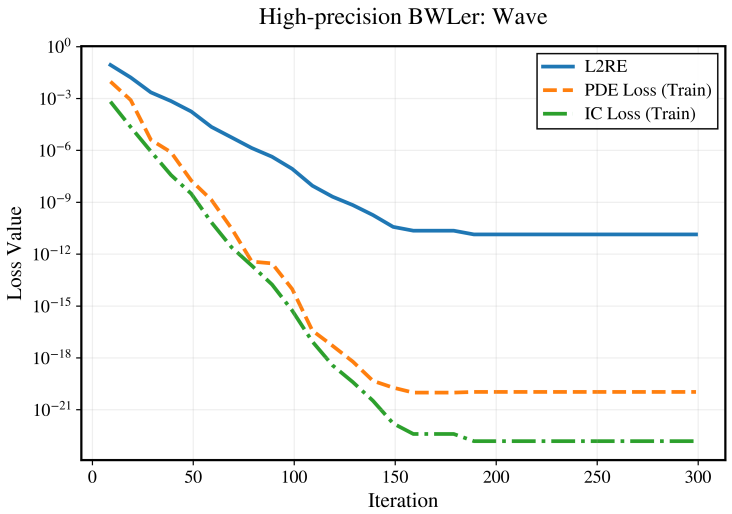

Figure 19: Loss curves for explicit BWLER trained with NNCG on wave equation (Table 2).

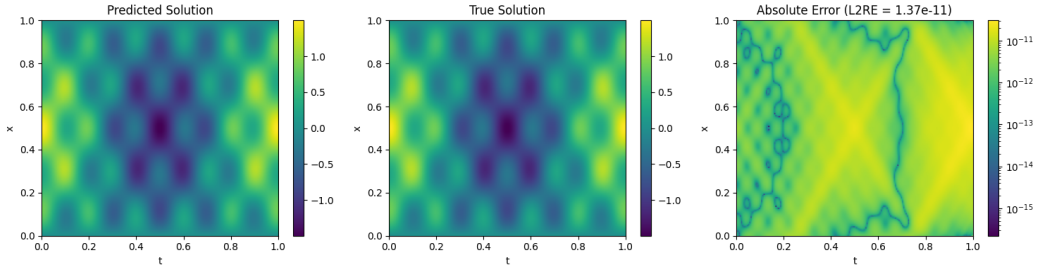

Figure 20: Explicit BWLER's learned solution and error residual on wave equation (Table 2).

Burgers' Equation.

- *Architecture.* We use $N_t = N_x = 321$, where we treat both time and space with a Chebyshev basis. For the experiments in Table 1, we use spectral derivatives in both space and time. For the experiment in Table 2, we use spectral derivatives in space, and finite difference derivatives in time, using 1st-order, 3-point finite difference stencils.
- *High-precision optimization.* We train with Nyström-Newton-CG for 850 steps, with a preconditioner rank of 1000 and 2000 CG iterations per step. On an A100, the total training takes about 8.2 hours (about 35 seconds per iteration).

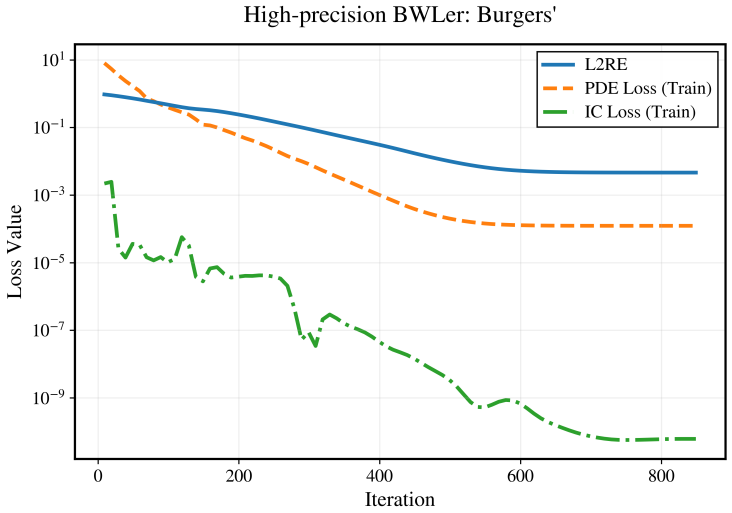

Figure 21: Loss curves for explicit BWLER trained with NNCG on Burgers' equation (Table 2).

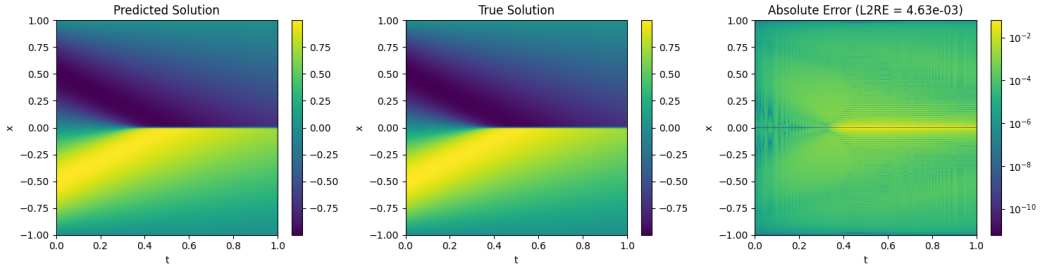

Figure 22: Explicit BWLER's learned solution and error residual on Burgers' equation (Table 2).

Poisson Equation.

- *Architecture.* We use $N_x = N_y = 51$, where we treat both dimensions with a Chebyshev basis.
- *High-precision optimization.* We train with Nyström-Newton-CG for 51,000 epochs with a preconditioner rank of 1000 and 64 CG iterations per step. On an A5000, the total training time is about 16 hours (about 1.2 seconds per iteration).

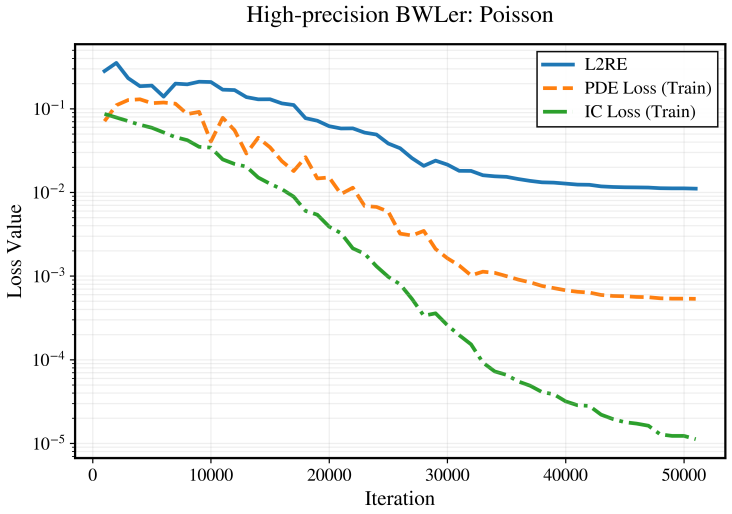

Figure 23: Loss curves for explicit BWLER trained with NNCG on Poisson equation (Table 2).

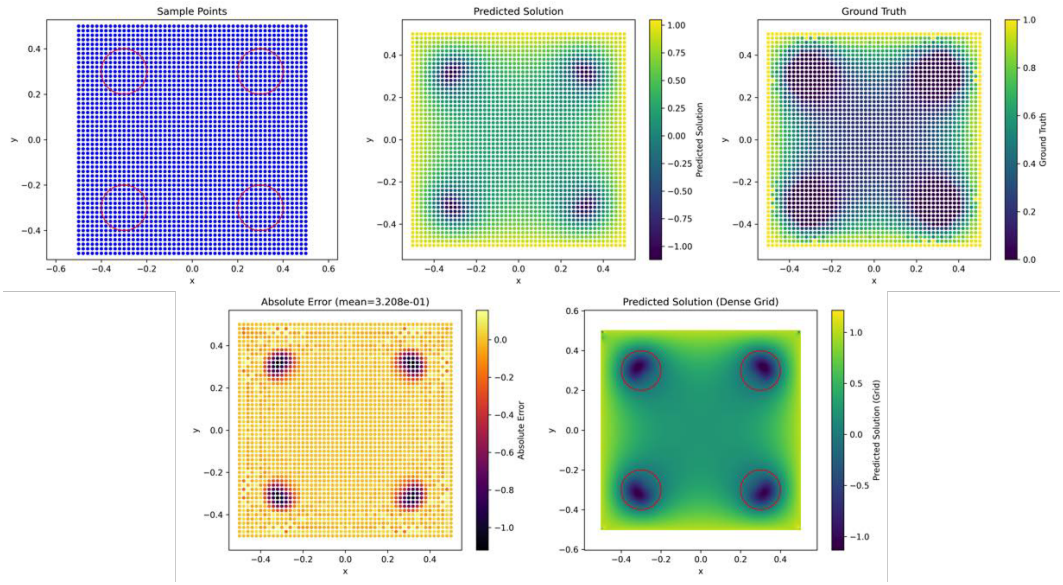

Figure 24: Explicit BWLER's learned solution and error residual on Poisson equation (Table 2).

D Theory

D.1 Formal statement and proof of Theorem 4.1

Definition D.1 (Interpolation and empirical Gram matrices)**.** Fix the Chebyshev–Gauss–Lobatto (CGL) nodes

$$x_j = \cos\big(j\pi/N\big), j = 0,...,N,$$

and let $\{\ell_j\}_{j=0}^N$ be the corresponding Lagrange basis polynomials [33], defined by:

$$\ell_j(x) = \prod_{\substack{0\le k\le N\\ k\neq j}} \frac{x-x_k}{x_j-x_k}, \qquad j = 0,...,N.$$

For any training set $\tilde{X} = \{\tilde{x}_i\}_{i=1}^M \subset [-1,1]$, the *interpolation matrix* $L \in \mathbb{R}^{M\times(N+1)}$ is

$$L_{i,j} = \ell_j(\tilde{x}_i).$$

Given a vector $f = \{f_j\}_{j=0}^N$ of function values at the Chebyshev nodes, the evaluations of the degree-N polynomial $f(x) = \sum_{j=0}^N f_j \ell_j(x)$ at the training points satisfy $\big[f(\tilde{x}_1),...,f(\tilde{x}_M)\big]^\top = Lf$.

We also define the *empirical value Gram matrix* as:

$$G_{\text{emp}}^M = \frac{1}{M} L^\top L.$$

Definition D.2 (Population (continuous) value Gram)**.** The *population Gram matrix* is

$$G_{\text{pop}} = \left[\int_{-1}^1 \ell_j(x)\ell_k(x)\,\frac{dx}{2}\right]_{j,k=0}^N.$$

By exactness of Clenshaw–Curtis quadrature on the CGL nodes [33], $G_{\text{pop}} = \text{diag}\big(w_0^{\text{CC}},...,w_N^{\text{CC}}\big)$, where:

$$w_j^{CC} = \begin{cases} \frac{\pi}{(2N)} & j = 0,N \\ \frac{\pi}{N} & j = 1,...,N-1, \end{cases}$$

so $\kappa^2(G_{\text{pop}}) = 2$.

Moreover, under uniform sampling, $G_{\text{emp}} \to G_{\text{pop}}$ (as $M\to\infty$) in spectral norm almost surely [35].

Lemma D.3 (Concentration of the empirical Gram)**.** *With Definitions D.1–D.2, fix* $0<\varepsilon<1$ *and* $\delta\in(0,1)$*. There is a universal constant* $C>0$ *such that if*

$$M \ge C\frac{(N+1)\log^2(N+1)\log\big((N+1)/\delta\big)}{\varepsilon^2},$$

then with probability at least $1-\delta$*,*

$$(1-\varepsilon)G_{\text{pop}} \preceq G_{\text{emp}}^M \preceq (1+\varepsilon)G_{\text{pop}}, \quad \kappa^2(G_{\text{emp}}^M) \le 2\frac{1+\varepsilon}{1-\varepsilon}.$$

Proof. Each row $u_i = (\ell_0(\tilde{x}_i),...,\ell_N(\tilde{x}_i))$ of L satisfies

$$\|u_i\|_2^2 \le (N+1)\Lambda_N^2,$$

where Λ_N, the *Lebesgue constant*, satisfies ([33, Thm. 16.1]):

$$\Lambda_N = \sup_{x\in[-1,1]} \sum_{j=0}^N |\ell_j(x)| = O(\log N).$$

Hence $\|u_i\|_2^2 \le C'(N+1)\log^2(N+1)$ for some constant C'. By the matrix–Bernstein inequality [34],

$$\|G_{\text{emp}} - G_{\text{pop}}\|_{\text{op}} \le \varepsilon$$

with probability $\ge 1-\delta$, provided $M \gtrsim (N+1)\log^2(N+1)\log((N+1)/\delta)/\varepsilon^2$. This implies $(1-\varepsilon)G_{\text{pop}} \preceq G_{\text{emp}} \preceq (1+\varepsilon)G_{\text{pop}}$, and hence $\kappa^2(G_{\text{emp}}) \le 2(1+\varepsilon)/(1-\varepsilon)$. □

We are now ready to state the full version of Theorem 4.1:

Theorem D.4 (Expressivity–optimization decomposition for interpolation with BWLER)**.** *Let $f:[-1,1]\to\mathbb{R}$ extend analytically to the Bernstein ellipse E_ρ with $\rho>1$ and write $M_f=\max_{z\in E_\rho}|f(z)|$. Fix training nodes $\tilde{X}=\{\tilde{x}_i\}_{i=1}^M\subset[-1,1]$ and let $L\in\mathbb{R}^{M\times(N+1)}$ be the interpolation matrix (Definition D.1). Define the condition number $\kappa^2(L)=\lambda_{\max}(L^\top L)/\lambda_{\min}(L^\top L)=\kappa(L)^2$ and the CGL Lebesgue constant $\Lambda_N=\sup_{x\in[-1,1]}\sum_{j=0}^N|\ell_j(x)|$.*

Initialize an (N+1)-parameter BWLER *with parameters $\theta^{(0)}=0$. Run t steps of gradient descent on the loss function $\mathcal{L}(\theta)=M^{-1}\|L\theta-f_{\tilde{X}}\|_2^2$ with optimal step–size $\eta=1/\lambda_{\max}(L^\top L)$. Denote the parameters of the t-th iterate* BWLER *polynomial by $\theta^{(t)}$ and the polynomial itself by $p_N^{(t)}:=p_N(x;\theta^{(t)})$. Then for* any *training set:*

$$\|f-p_N^{(t)}\|_\infty\le\underbrace{\frac{2M_f}{\rho^N-1}}_{\textit{expressivity}}+\underbrace{\|\theta^\star\|_2\Lambda_N\exp\big(-t/\kappa^2(L)\big)}_{\textit{optimization}}\qquad(\dagger)$$

where $\theta^\star=(f(x_0),\dots,f(x_N))^\top$ interpolates f on the CGL nodes.

Proof. Gradient descent on the quadratic loss function yields $\|\theta^{(t)}-\theta^\star\|_2\le e^{-t/\kappa^2(L)}\|\theta^\star\|_2$ [5]. For any $x\in[-1,1]$:

$$\begin{aligned}|p_N^{(t)}-p_N^*|&=\Big|\sum_{j=0}^N\big(\theta_j^{(t)}-\theta_j^\star\big)\ell_j(x)\Big|\\&\le\Big(\sum_{j=0}^N|\ell_j(x)|\Big)\|\Delta\theta\|_\infty\\&\le\Lambda_N\|\Delta\theta\|_2\\&\le\Lambda_N\|\theta^\star\|_2e^{-t/\kappa^2(L)}.\end{aligned}$$

Taking the supremum in x and adding the standard Bernstein-ellipse expressivity bound (Theorem 2.2) finishes the proof. □

Corollary D.5 (Uniformly sampled nodes)**.** *Let $0<\varepsilon<\frac12,\delta\in(0,1)$ and draw $\tilde{X}$ uniformly without replacement from $[-1,1]$. If*

$$M\ge C(N+1)\log^2(N+1)\log\big((N+1)/\delta\big)/\varepsilon^2,$$

then by Lemma D.3, with probability $\ge1-\delta$:

$$\kappa^2(L)\le2(1+\varepsilon)/(1-\varepsilon).$$

Inserting this in Equation (†) *gives*

$$\|f-p_N^{(t)}\|_\infty\le\frac{2M_f}{\rho^N-1}+\Lambda_N\|\theta^\star\|_2\exp\Big(-t\tfrac{1-\varepsilon}{2(1+\varepsilon)}\Big).$$

Intuitively, Theorem D.4 and Corollary D.5 capture a precision-conditioning tradeoff involving N and M:

- N **too small (high bias).** The expressivity term dominates, and error convergence is exponential (dependent on target function smoothness) as N increases.
- $N\ll M$ **but large.** The empirical Gram is well conditioned ($\kappa^2(L)=O(1)$) as soon as $M\gtrsim N\log^2N$, so the training gap decreases exponentially.
- $N+1=M$ **random sampling (poor conditioning).** When $M=N+1$ and the points $\{\tilde{x}_i\}$ are drawn arbitrarily, L is square and generically invertible but has a condition number that grows rapidly with N. As a result, gradient descent converges only at rate $\exp\big(-t/\kappa^2(L)\big)$ with $\kappa^2(L)\gg1$, so achieving small training error requires a long training time, $t\gg\kappa^2(L)$.

D.2 Formal statement and proof of Theorem 5.1

We make the simplifying assumption that our collocation points for the PINN loss are chosen to be the same Chebyshev–Gauss–Lobatto (CGL) nodes

$$x_j = \cos(j\pi/N), j = 0,...,N$$

as used in the BWLER parameterization. (This is the "fixed nodal collocation" scheme we describe in Appendix B.) Define the Lagrange basis of the CGL nodes $\{\ell_j\}_{j=0}^N$, and let

$$\Lambda_N = \sup_{x\in[-1,1]} \sum_{j=0}^{N} |\ell_j(x)| = O(\log N)$$

be the Lebesgue constant [33].

Definition D.6 (Collocation matrix for a PDE)**.** Given a linear differential operator

$$L = \sum_{\alpha=0}^{d} a_\alpha(x)\partial_x^\alpha \tag{8}$$

and its numerical surrogate $\widetilde{L}$, define the square collocation matrix

$$\widetilde{A}_{i,j} = (\widetilde{L}\ell_j)(x_i), \qquad \widetilde{b}_i = g(x_i)\,(= Lu(x_i)),$$

where the collocation points are the *same* CGL nodes x_i. Also let $\kappa^2(\widetilde{A}) = \lambda_{\max}(\widetilde{A}^\top\widetilde{A})/\lambda_{\min}(\widetilde{A}^\top\widetilde{A})$.

Definition D.7 (Operator mis-specification)**.** For polynomials v of degree $\le N$, define:

$$\varepsilon_{\rm op}(N) := \sup_{\deg(v)\le N,\, \|v\|_\infty \le 1} \|(L-\widetilde{L})v\|_\infty.$$

Intuitively, $\epsilon_{op}(N)$ represents the worst-case bias introduced by replacing the true differential operator L with its numerical surrogate $\widetilde{L}$.

We are now ready to state the full version of Theorem 5.1:

Theorem D.8 (Expressivity–bias–optimization decomposition for PDE learning with BWLER)**.** *Let $u:[-1,1]\to\mathbb{R}$ solve $Lu = g$ with analytic data and extend analytically to the Bernstein ellipse E_ρ ($\rho>1$); set $M_u = \max_{z\in E_\rho}|u(z)|$. Form the collocation system $(\widetilde{A},\widetilde{b})$ from Definition D.6.*

Initialize an $(N+1)$-parameter BWLER *with parameters $\theta^{(0)} = 0$ and run t steps of gradient descent on the quadratic loss $\mathcal{L}(\theta) = \frac{1}{N+1}\|\widetilde{A}\theta - \widetilde{b}\|_2^2$ using the optimal step size $\eta = 1/\lambda_{\max}(\widetilde{A}^\top\widetilde{A})$. Denote the parameters of the t-th iterate by $\theta^{(t)}$ and the resulting polynomial by*

$$u_N^{(t)}(x) = \sum_{j=0}^{N} \theta_j^{(t)} \ell_j(x).$$

Moreover define

$$u^*(x) := \sum_{j=0}^{N} u(x_j)\ell_j(x), \qquad \widetilde{u}(x) := \sum_{j=0}^{N} \theta_j^* \ell_j(x),$$

where $\theta^* = \mathrm{argmin}_\theta \|\widetilde{A}\theta - \widetilde{b}\|_2$. Then, when the collocation points coincide with the CGL grid:

$$\|u - u_N^{(t)}\|_\infty \le \underbrace{\frac{2M_u}{\rho^N - 1}}_{\textit{expressivity}} + \underbrace{M_u\Lambda_N\varepsilon_{\rm op}(N)}_{\textit{bias/misspecification}} + \underbrace{\Lambda_N\|\theta^*\|_2 \exp\!\big(-t/\kappa^2(\widetilde{A})\big)}_{\textit{optimization}}. \tag{9}$$

Proof. **1. Expressivity term.** This term accounts for the gap between the true solution to the true PDE, u, and the best polynomial approximation to it, u^*. Let u^* be the degree-N interpolant of the true solution on CGL. Then Theorem 2.2 yields: $\|u - u^*\|_\infty \le 2M_u/(\rho^N - 1)$.

2. Bias/misspecification term. This term accounts for the gap between the best polynomial approximation to the PDE solution, u^*, and the true solution to the numerical surrogate, $\widetilde{u}$. At each node,

$$r_i = (\widetilde{L} - L)u^*(x_i), \qquad |r_i| \le M_u \varepsilon_{\mathrm{op}}(N).$$

Hence $\|\widetilde{A}\theta^* - \widetilde{b}\|_\infty = \max_i |r_i|$, and interpolating these residuals off the grid gives

$$\|\widetilde{u} - u^*\|_\infty \le \Lambda_N \max_i |r_i| \le \Lambda_N M_u \varepsilon_{\mathrm{op}}(N).$$

3. Optimization term. This term accounts for the gap between the t-th iterate, $u_N^{(t)}$, and the true solution to the numerical surrogate PDE, $\widetilde{u}$. Gradient descent on the quadratic loss function yields

$$\|\theta^{(t)} - \theta^*\|_2 \le \exp\big(-t/\kappa^2(\widetilde{A})\big)\|\theta^*\|_2.$$

For any x,

$$|u_N^{(t)} - \widetilde{u}(x)| = \Big|\sum_{j=0}^{N} (\theta_j^{(t)} - \theta_j^*)\ell_j(x)\Big| \le \Lambda_N \|\theta^{(t)} - \theta^*\|_2,$$

so $\|\widetilde{u} - u_N^{(t)}\|_\infty \le \Lambda_N \|\theta^*\|_2 e^{-t/\kappa^2(\widetilde{A})}$.

Combining the three bounds yields (9). □

Corollary D.9 (Finite–difference surrogate of order k)**.** *Let $\widetilde{L}$ replace each d-th derivative in L by a k-th-order finite–difference stencil on the CGL grid [13]. Then*

$$\varepsilon_{\mathrm{op}}(N) = O\big(N^{-(k+1-d)}\big),$$

so Theorem D.8 yields

$$\|u - u_N^{(t)}\|_\infty \le \frac{2M_u}{\rho^N - 1} + \widetilde{O}\Big(N^{-(k+1-d)}\Big) + \Lambda_N \|\theta^\star\|_2 \exp\big(-t/\kappa^2(\widetilde{A})\big).$$

Intuitively, Theorem D.8 and Corollary D.9 capture a precision–conditioning tradeoff involving the accuracy of the derivative approximation:

- N **too small (low precision ceiling).** The expressivity term $\dfrac{2M_u}{\rho^N - 1}$ dominates. Even though error decays exponentially in N, we use too few polynomial basis elements to resolve the solution's high-frequency features.
- **Low–order finite differences (low precision ceiling, faster convergence).** If $\widetilde{L}$ uses a kth–order stencil with $k+1-d$ small, then the bias term

  $$\text{bias} = M_u \Lambda_N \varepsilon_{\mathrm{op}}(N) = O\big(N^{-(k+1-d)} \log N\big),$$

  decays only algebraically and dominates.
- **Spectral collocation (high precision ceiling, slower convergence).** With $\widetilde{L} \approx L$ and large N, the optimization term $\Lambda_N \|\theta^*\|_2 e^{-t/\kappa^2(A)}$ dominates. For a d-th order operator the CGL collocation matrix has $\kappa^2(A) = O(N^{2d})$ [32], so GD converges at rate $\exp\big(-t/O(N^{2d})\big)$, requiring $t \gg N^{2d}$ iterations.