ADAPTIVE BACKPROPAGATION FOR STABLE PHYSICS-INFORMED NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Physics-informed neural networks (PINNs) differentiate losses containing spatial or temporal derivatives of network outputs, making backpropagation expensive and potentially unstable. We study a semi-gradient path that preserves residual values while replacing the residual Jacobian with the network-output Jacobian. This reduces update cost and can improve numerical stability, but prolonged use may cause oscillations and limit accuracy. We propose Prediction-guided Adaptive Semi-gradient Switching (PASS), which starts with semi-gradients and switches once to full gradients when successive prediction changes on a fixed probe set lose positive alignment. The criterion requires only output evaluations and no reference solution. Experiments demonstrate improved accuracy across optimizers, architectures, and PINN training techniques, at computational cost comparable to full-gradient training.

1 INTRODUCTION

Physics-informed neural networks (PINNs) approximate solutions to partial differential equations (PDEs) by minimizing equation residuals together with boundary and initial-condition penalties (Raissi et al., 2019; Karniadakis et al., 2021). Their formulation incorporates physical constraints directly into training, but also introduces derivative computations beyond those in standard supervised learning. Evaluating a PDE residual requires spatial or temporal derivatives of the network output. Updating the parameters then requires differentiating through these quantities. For example, a residual containing u_{xx} requires both evaluating this second spatial derivative and computing its parameter sensitivity during backpropagation. These nested computations can be expensive (He et al., 2023; Cho et al., 2023). Coordinate derivatives can also vanish or explode in deep networks (Meng et al., 2026), motivating closer examination of how the residual's derivative structure enters parameter updates.

An alternative is to use the current PDE residual as a training signal without differentiating through its full computation. The gradient-residual (GR) construction of Cao & Zhang (2026) follows this principle by replacing the residual Jacobian with the network-output Jacobian in the PDE update. We refer to this update as a semi-gradient path and implement it using stop-gradient, while retaining ordinary gradients for boundary and initial-condition penalties. The evaluated PDE residual is unchanged, but its parameter sensitivities are excluded from the update. Thus, the coordinate derivatives required to evaluate the PDE are still computed, while their subsequent parameter differentiation is avoided in the PDE backward path.

The full and semi paths have different computational and optimization properties. For a symmetric positive-definite linear system optimized directly in solution space, the semi direction avoids the squared condition number associated with residual least squares (Cao & Zhang, 2026). In neural PDE solvers, however, replacing the residual Jacobian changes the update direction, which need not descend the original residual loss. Our experiments illustrate this trade-off. On the biharmonic problem, semi updates are cheaper and remain finite over a wider tested learning-rate range than full updates. On Burgers, semi training can reduce solution error rapidly at first, but continued use produces oscillations and worse final accuracy than full training. These observations suggest that a useful backward path can depend on the stage of training.

Stabilized gradient residual (SGR) training combines the GR and full residual directions using a fixed weight (Cao & Zhang, 2026). This provides a way to balance their behavior, but retains full residual

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

differentiation at every update. We instead investigate a sequential strategy: use semi-gradients initially, then restore full gradients for subsequent refinement. This raises a practical question: when should the transition occur? A fixed switching time must be selected in advance, although training trajectories can differ across PDEs, optimizers, and network architectures. An adaptive strategy should use information available during training without requiring a reference solution or repeatedly computing both gradients.

We propose **Prediction-guided Adaptive Semi-gradient Switching (PASS)**, which makes a one-time transition based on changes in the predicted solution. PASS periodically evaluates the network on a fixed probe set and compares prediction displacements over two consecutive intervals. It switches to full gradients when their cosine similarity first becomes nonpositive. This criterion detects a loss of directional persistence in the predicted field using only output evaluations. It serves as a heuristic for deciding when to restore residual sensitivities; it does not estimate solution error or guarantee descent. After the transition, training continues with full gradients while preserving the optimizer state and learning-rate schedule.

Our evaluation separates the numerical and computational properties of the two paths from the effectiveness of switching between them. The biharmonic experiment examines stability and cost, while Burgers and reaction–diffusion test final accuracy and compatibility with different training configurations. The fixed-switch comparison further asks whether prediction-guided timing improves on predetermined transitions. The contributions are as follows:

1. **Characterization of residual backward paths.** We examine the derivative computations involved in full residual backpropagation and compare them with semi-gradient updates. Controlled experiments identify computational savings and improved numerical behavior in the tested settings, together with limitations of continued semi-gradient training.
2. **Prediction-guided adaptive switching.** We introduce PASS, which uses changes in network predictions to select a one-time transition from semi to full gradients. The decision requires neither a reference solution nor full-gradient evaluation during monitoring, and preserves the existing optimizer state and schedule.
3. **Evaluation across PINN training configurations.** We compare PASS with fixed paths, gradient mixtures, and switching schedules, and assess its compatibility with different optimizers, architectures, adaptive sampling, and loss weighting.

2 PRELIMINARIES

2.1 PHYSICS-INFORMED NEURAL NETWORKS

Consider a scalar PDE on a domain $\Omega \subset \mathbb{R}^d$, with prescribed conditions on $\Gamma \subseteq \partial\Omega$:

$$\mathcal{N}[u^\star](x) = f(x), \quad x \in \Omega, \qquad \mathcal{C}[u^\star](x) = g(x), \quad x \in \Gamma.$$

Here $u^\star$ is the unknown solution, $\mathcal{N}$ is the PDE differential operator, $\mathcal{C}$ specifies the boundary or initial conditions, and f and g are given functions. For evolution problems, x includes both spatial and temporal coordinates, and Γ includes the initial-time surface and the prescribed spatial boundary. A physics-informed neural network (PINN) approximates $u^\star$ by $u_\theta(x)$ and trains its parameters θ to satisfy the PDE and these conditions (Raissi et al., 2019).

PDE residual and residual loss. The pointwise PDE residual is $r_\theta(x) = \mathcal{N}[u_\theta](x) - f(x)$. It measures the mismatch in the governing equation and vanishes wherever the network satisfies that equation. Evaluating $r_\theta(x)$ requires derivatives of $u_\theta(x)$ with respect to the coordinates x, as specified by $\mathcal{N}$. These spatial derivatives, and temporal derivatives for evolution problems, are computed by automatic differentiation. For example, a PDE containing a second spatial derivative requires evaluating $\partial^2 u_\theta / \partial x_k^2$ as part of the residual, where x_k is a spatial coordinate.

At interior collocation points $X = \{x_i\}_{i=1}^n$, we collect the residual values in $r_\theta = (r_\theta(x_i))_{i=1}^n \in \mathbb{R}^n$. The PDE residual loss is

$$L_r(\theta) = \frac{1}{n} \sum_{i=1}^{n} r_\theta(x_i)^2 = \frac{1}{n} \|r_\theta\|_2^2. \tag{1}$$

We similarly write $u_\theta = (u_\theta(x_i))_{i=1}^n \in \mathbb{R}^n$ for the network-output vector at the same points. Standard PINN training differentiates L_r with respect to θ, including the parameter dependence of the coordinate derivatives used to evaluate the residual. Coordinate differentiation therefore enters loss evaluation, while parameter differentiation determines the training update.

Constraint loss and training objective. We collect boundary and initial-condition penalties in $B(\theta)$. For example, a condition sampled at $X_b = \{x_j^b\}_{j=1}^{n_b} \subset \Gamma$ contributes $\frac{\lambda_b}{n_b}\sum_{j=1}^{n_b} \|\mathcal{C}[u_\theta](x_j^b) - g(x_j^b)\|_2^2$, where $\lambda_b > 0$ is its weight. Conditions enforced exactly by the network construction require no additional penalty. The training objective is

$$L(\theta) = L_r(\theta) + B(\theta) = \frac{1}{n}\|r_\theta\|_2^2 + B(\theta). \tag{2}$$

Our focus is the parameter differentiation of L_r; constraint penalties retain their ordinary gradients.

2.2 Stop-gradient

A stop-gradient operation, denoted by $\mathrm{stopgrad}$, preserves its argument's value while blocking differentiation through that branch of the computation graph. For a parameter-dependent quantity z_θ,

$$\mathrm{stopgrad}(z_\theta) = z_\theta \quad \text{(forward)}, \qquad \frac{\partial\ \mathrm{stopgrad}(z_\theta)}{\partial\theta} = 0 \quad \text{(backward)}. \tag{3}$$

The backward equality specifies an automatic-differentiation rule. The argument is recomputed at each evaluation, so its numerical value can change as the parameters change. Stop-gradient thus allows a quantity to contribute to the evaluated loss without propagating derivatives through its computational branch.

An ordinary stop-gradient operation blocks derivatives through its argument with respect to all upstream variables, including coordinates and parameters. Its placement therefore matters in PINNs. In Section 3, we first compute the coordinate derivatives needed to evaluate the PDE residual, then apply stop-gradient to construct an alternative parameter-update direction. This preserves the residual's forward value while changing how the residual loss is differentiated during training.

3 Methodology

3.1 Analysis of residual backpropagation

PINN training involves two distinct differentiation tasks. Evaluating the PDE residual requires derivatives of $u_\theta(x)$ with respect to the coordinates x. Computing the training gradient then requires differentiating the residual with respect to θ, including the parameter dependence of those coordinate derivatives. Coordinate derivatives can already vanish or explode at initialization, as observed in the depth-dependent MLP and ResNet experiments of Meng et al. (2026). We examine how the factors involved in their evaluation also enter residual backpropagation.

Consider a smooth network with $h_0 = x$, $h_\ell = F_\ell(h_{\ell-1};\theta_\ell)$, and $u_\theta = h_L$. Let $J_\ell = \partial h_\ell/\partial h_{\ell-1}$ denote the Jacobian of layer ℓ. The first coordinate derivative and its derivative with respect to a parameter coordinate θ_a satisfy

$$\frac{\partial u_\theta}{\partial x} = J_L \cdots J_1, \qquad \frac{\partial}{\partial\theta_a}\left(\frac{\partial u_\theta}{\partial x}\right) = \sum_{\ell=1}^{L} J_L \cdots J_{\ell+1}\frac{dJ_\ell}{d\theta_a}J_{\ell-1}\cdots J_1. \tag{4}$$

Here $dJ_\ell/d\theta_a$ includes dependence through preceding activations at fixed x, and empty products denote identity maps. The expansion shows that layer Jacobians used to evaluate the coordinate derivative also appear in its parameter sensitivity. Higher-order PDE derivatives introduce further sums and products through repeated applications of the chain rule.

The squared residual introduces an additional multiplication, $\partial_{\theta_a}(r_\theta^2) = 2r_\theta\partial_{\theta_a}r_\theta$. Thus, derivative-dependent residual values multiply their parameter sensitivities in the update. These computations provide possible routes for numerical attenuation or amplification. The actual gradient scale also depends on the differentiated factors and cancellations, so the expansion does not imply that small or

large coordinate derivatives necessarily produce small or large parameter gradients. Differentiating through the residual additionally incurs the cost of mixed coordinate– parameter derivatives, which contributes to the computational burden of PINN training (He et al., 2023).

We investigate numerical stability on the fourth-order biharmonic problem in Section 4.1. With the same model and a budget of 10,000 SGD updates, full residual backpropagation completes only 9 of 18 runs across six learning rates and three seeds (Table 7). At learning rate 3×10^{-4}, all three runs become nonfinite between updates 764 and 823. These observations motivate examining whether a simpler residual backward path can improve numerical behavior and reduce computational cost.

3.2 Semi-gradient backpropagation for PINNs

We consider a semi-gradient construction that preserves the residual value while replacing its parameter Jacobian by the network-output Jacobian. At the interior collocation points, define

$$\widetilde{r}_\theta = \mathrm{stopgrad}(r_\theta - u_\theta) + u_\theta. \tag{5}$$

The coordinate derivatives required by the PDE are computed before applying stop-gradient. The forward value of $\widetilde{r}_\theta$ equals r_θ, so substituting it into (2) preserves the evaluated loss. During backpropagation, the detached term contributes no parameter derivative. Writing $J_r = \partial r_\theta/\partial\theta$ and $J_u = \partial u_\theta/\partial\theta$, the full and semi update directions are

$$g_F = \frac{2}{n} J_r^\top r_\theta + \nabla_\theta B, \qquad g_S = \frac{2}{n} J_u^\top r_\theta + \nabla_\theta B. \tag{6}$$

This replaces the entire residual Jacobian, including sensitivities of both differential and algebraic PDE terms. The residual remains the driving signal, but its backward path passes only through the network output. Boundary and initial-condition penalties retain their ordinary gradients.

The semi PDE direction is proportional to the gradient-residual (GR) direction of Cao & Zhang (2026), who combine it with the full residual gradient in stabilized gradient residual (SGR) training. Equation (5) implements this direction through a modified residual. Our focus is on its numerical and computational properties and on deciding when to restore full residual differentiation.

To illustrate a possible conditioning benefit, consider a linear PDE discretization $Au^\star = f$, where A is symmetric positive definite, and optimize u directly without additional constraints. For $e = u - u^\star$, the full and semi directions are $2A^2e/n$ and $2Ae/n$, respectively. The full direction is the gradient of the residual loss $e^\top A^2 e/n$, whereas the semi direction is the gradient of the alternative potential $e^\top Ae/n$. Their Hessians are $2A^2/n$ and $2A/n$, with condition numbers $\kappa(A)^2$ and $\kappa(A)$ (Cao & Zhang, 2026). This reduction improves the worst-case convergence rate of gradient descent with a suitable step size (Boyd & Vandenberghe, 2004). It illustrates a benefit in solution space; it does not establish the same conditioning improvement under nonlinear neural parameterization or for general PDE operators.

The experiments show practical differences between the two paths. Figure 1 compares PDE-gradient norms at identical parameters along a semi trajectory on Burgers. The median full-to-semi ratio is 36.4, indicating a substantial scale difference in this run. On the biharmonic problem, semi completes all 18 runs in the shared learning-rate sweep, compared with nine finite completions for full (Table 1). At the common finite rate 10^{-5}, semi reduces total training time from 73.5 to 31.3 seconds (Table 2). These results demonstrate lower cost and a wider tested range of learning rates without nonfinite failures. Semi still requires finite coordinate derivatives and residual values, and derivative-based constraint penalties can retain mixed coordinate–parameter differentiation through B.

The replacement also changes the optimization dynamics. In general, $-g_S$ need not be a descent direction for the original objective. For an infinitesimal SGD step,

$$\frac{d}{d\eta} L(\theta - \eta g_S)\bigg|_{\eta=0} = -\langle g_F, g_S \rangle. \tag{7}$$

Strict first-order descent therefore requires $\langle g_F, g_S \rangle > 0$. If the constraint gradient vanishes, semi stationarity requires $J_u^\top r_\theta = 0$, whereas full stationarity requires $J_r^\top r_\theta = 0$. Both can occur with nonzero residuals, but the two conditions need not coincide. The residual sign convention also matters: replacing r_θ by $-r_\theta$ leaves the full squared-residual gradient unchanged but reverses the semi PDE contribution.

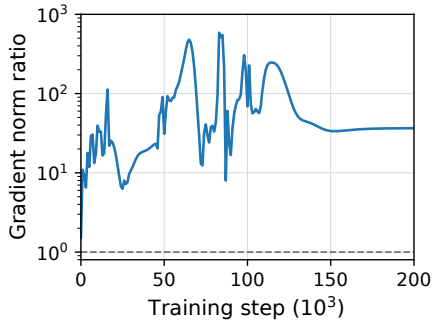

Figure 1: Full/semi PDE-gradient norm ratio at identical parameters along the semi trajectory. The median over 201 states is 36.4; the dashed line marks equal norms.

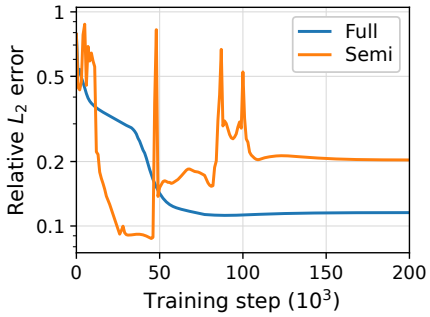

Figure 2: Training dynamics on Burgers. Full and semi use the same initialization, training points, and SGD settings.

Figure 2 illustrates the resulting trade-off in a Burgers run. With matched initialization, training points, and SGD settings, semi initially reaches a lower solution error, but later oscillates and finishes above full. Together with its lower update cost, this observation motivates using semi for initial progress and restoring full differentiation for subsequent refinement. The favorable duration of the semi phase is problem dependent, motivating an adaptive decision.

3.3 Prediction-guided adaptive switching

We propose **Prediction-guided Adaptive Semi-gradient Switching (PASS)**, which starts with semi-gradient updates and uses changes in the predicted solution to determine a one-time transition to full gradients. The aim is to retain useful initial progress from the semi path while restoring residual sensitivities when its output trajectory loses directional persistence.

Let K be an observation interval, and fix a probe set X_p to the initial interior collocation points. We record predictions every K updates and compare the changes over consecutive intervals:

$$z_j = u_{\theta_{jK}}(X_p), \qquad \delta_j = z_j - z_{j-1}, \qquad c_j = \frac{\langle \delta_j, \delta_{j-1} \rangle}{\|\delta_j\|_2 \|\delta_{j-1}\|_2}, \quad j \geq 2, \tag{8}$$

where δ_j describes the change in the predicted solution over one observation interval. Positive c_j indicates that successive prediction changes are aligned, and PASS continues with semi-gradients. At the first observation with $c_j \leq 0$ and updates remaining, PASS switches permanently to full gradients from the next update onward. For finite displacements, we set $c_j = 0$ if either displacement is zero. The earliest possible switch is after $2K$ updates. If the event never occurs, training remains semi. The switch preserves the optimizer state and global learning-rate schedule. Algorithm 1 summarizes the procedure; Appendix B.2 specifies numerical edge cases.

Prediction-guided Adaptive Semi-gradient Switching (PASS)

1. Initialize the network. Fix the update budget T, observation interval K, and probe set X_p. Record $z_0 = u_{\theta_0}(X_p)$, set $j = 0$, and start with the semi path.
2. Perform the next K semi updates, or stop at budget T. If updates remain, increment j, record z_j, and form $\delta_j = z_j - z_{j-1}$.
3. If $j \geq 2$, compute c_j using (8), assigning zero when either finite displacement is zero. If $c_j \leq 0$, switch to full for all remaining updates and stop observing.
4. Otherwise, repeat steps 2–3.

Algorithm 1: PASS starts with semi-gradient updates and switches once when successive prediction changes cease to reinforce one another.

We monitor predictions because their changes describe motion of the approximated solution at fixed locations. A scalar loss aggregates residuals and does not retain this directional information. Directly

testing the alignment in (7) would require computing the full residual gradient during the semi phase. In contrast, the output cosine requires only periodic network evaluations. Keeping X_p fixed makes the displacements comparable even when adaptive sampling changes the training points.

The criterion is inspired by the use of directional information in adaptive restart methods (O'Donoghue & Candès, 2015). A nonpositive cosine identifies a turn in the sampled output trajectory, which we use as an empirical signal to restore full differentiation. It does not estimate solution error, test full/semi gradient alignment, or guarantee that the switch improves the objective. Its usefulness is evaluated in Sections 4.2 and 4.3.

The one-time transition also avoids combining both backward paths at every update. A fixed combination such as SGR retains residual differentiation throughout training. A smaller full-gradient weight can moderate large finite contributions, but it does not remove that computation or prevent already nonfinite gradient entries from entering the combined update. PASS omits this path from the PDE update during the semi phase and restores it after the switch. Whether the subsequent full phase remains numerically stable is an empirical property of the training configuration, rather than a guarantee of the cosine criterion.

For weighted residual losses, we apply the same Jacobian replacement with the weights held fixed during differentiation. Auxiliary training procedures retain their own computations. In particular, the NTK reweighting experiment periodically evaluates full residual Jacobians even during the semi phase, as detailed in Appendix C.3.

PASS requires periodic prediction evaluations and the vector operations in (8), which stop after switching. We use $K = 500$ for Burgers and $K = 1,000$ for reaction–diffusion. The studies in Sections 4.1 and 4.2 use SGD without momentum to isolate backward-path effects. Section 4.3 evaluates compatibility with Adam, alternative architectures, and other PINN training methods.

4 Experiments

We examine the numerical stability of semi-gradient backpropagation on the biharmonic equation, then evaluate path switching on Burgers and reaction–diffusion.

Setup and comparisons. All experiments use JAX/Flax (Bradbury et al., 2018; Heek et al., 2026) in float64 on RTX 4090 GPUs with three seeds. Within each setting, methods share initialization, initial training points, optimizer, learning-rate schedule, and update budget; points remain fixed except for RAD. Accuracy is measured by $E_{\mathrm{rel}} = \|u_\theta(X_e) - u^\star(X_e)\|_2 / \|u^\star(X_e)\|_2$. We compare **Full**, **Semi**, **PASS**, and a PDE-only **SGR** baseline (Cao & Zhang, 2026) using its default $\omega = 0.9$; SGR implementation details are given in Appendix A.2.

4.1 Biharmonic equation

To evaluate the strengths and limitations of semi-gradient and full-gradient training, we compare their numerical stability, accuracy, and computational cost on the biharmonic test problem considered by Li et al. (2022):

$$\Delta^2 u = 4\pi^4 \sin(\pi x)\sin(\pi y), \qquad (x, y) \in [0, 1]^2. \tag{9}$$

We impose $u = \Delta u = 0$ on the boundary, with exact solution $u^\star = \sin(\pi x)\sin(\pi y)$. The fourth-order residual provides a test of numerical instability along the backward path.

We use a four-layer, width-64 tanh MLP and 10,000 momentum-free SGD updates at each constant learning rate. The factor $x(1-x)y(1-y)$ enforces $u = 0$, while the Laplacian constraint remains a soft loss. Full, Semi, and SGR share the same initialization and fixed training points for each paired run. The complete eight-rate sweep is reported in Table 7.

Semi remains numerically stable over a wider learning-rate range than Full. Across the six rates in Table 1, Semi completes all 18 runs, compared with nine for Full and 15 for SGR. Full encounters nonfinite values in every run at 3×10^{-4} and above, preventing optimization from completing. At the commonly used PINN learning rate 10^{-3}, all three Full runs fail, whereas Semi trains stably and reaches a mean relative L_2 error of 3.64×10^{-3} (Table 7). SGR tolerates larger rates than Full but still fails at 3×10^{-3}, showing that a weighted full-gradient contribution does not eliminate these

Table 1: Finite runs on the biharmonic equation (9) across learning rates. Each entry counts runs completing 10,000 SGD updates without nonfinite training values, out of three random seeds.

LR	Full	Semi	SGR
10^{-5}	3/3	3/3	3/3
3×10^{-5}	3/3	3/3	3/3
10^{-4}	3/3	3/3	3/3
3×10^{-4}	0/3	3/3	3/3
10^{-3}	0/3	3/3	3/3
3×10^{-3}	0/3	3/3	0/3

Table 2: Relative L_2 error and training cost on the biharmonic equation (9) after 10,000 SGD updates. Errors give mean $\pm$ SD; costs give mean seconds over three random seeds. NaN marks numerical failure; † marks time to failure. Lower errors are bold when both methods finish.

	Relative L_2 ($\times 10^{-3}$)		Cost (s)	
LR	Full	Semi	Full	Semi
3×10^{-4}	NaN	1.74 ± 0.43	$11.4^{\dagger}$	31.2
10^{-4}	297.5 ± 24.5	$\mathbf{4.82 \pm 2.62}$	73.5	31.2
3×10^{-5}	97.4 ± 165.6	$\mathbf{22.5 \pm 6.2}$	73.4	31.3
10^{-5}	$\mathbf{2.62 \pm 0.80}$	207.4 ± 10.2	73.5	31.3

numerical failures. This difference is consistent with the backward path analysis in Section 3.1. PASS remains on the semi path throughout training and matches Semi's solution errors.

Full can achieve higher accuracy when training remains numerically stable, whereas Semi is better suited to rapid convergence during early training with larger learning rates. As shown in Table 2, reducing the learning rate improves Full's accuracy, allowing it to outperform Semi at the smallest rate shown. Semi, however, can exploit larger stable learning rates to obtain accurate solutions at lower computational cost. At the rates yielding each method's lowest error in this comparison, Semi also reaches the same accuracy target in fewer updates and less time (Appendix A.3). These findings support PASS's strategy of using semi-gradients for rapid, stable initial training and switching to full gradients for later refinement. This design combines the strengths of the two backward paths while mitigating their respective limitations. The following Burgers experiment demonstrates how this transition combines fast initial convergence with improved final accuracy.

4.2 Burgers equation

We use Burgers to test whether an initial semi phase followed by full differentiation can improve final accuracy. We consider the viscous Burgers benchmark used by Raissi et al. (2019):

$$u_t + uu_x - \frac{0.01}{\pi} u_{xx} = 0, \qquad (x, t) \in [-1, 1] \times [0, 1], \tag{10}$$

with $u(x, 0) = -\sin(\pi x)$ and zero spatial boundary values.

All methods use a four-layer, width-64 tanh MLP and momentum-free SGD for 200,000 updates, with base rate 3×10^{-3}, 2,000-step warmup, and cosine decay. We additionally test fixed Semi-to-Full handoffs at 10%, 25%, 50%, and 75% of training. Only the 75% handoff remains finite across all three seeds, so we report it as the fixed-switch baseline.

Table 3: **Burgers accuracy at a matched update budget.** Final relative L_2 error and training time after 200,000 updates.

Method	Relative L_2	Time (s)
Full	$(9.36 \pm 2.37) \times 10^{-2}$	489.8 ± 0.4
Semi	$(1.43 \pm 0.55) \times 10^{-1}$	259.5 ± 0.4
SGR	$(1.47 \pm 0.75) \times 10^{-1}$	540.8 ± 0.2
Fixed (75%)	$(3.31 \pm 1.91) \times 10^{-1}$	322.6 ± 2.9
PASS	$\mathbf{(5.74 \pm 1.73) \times 10^{-2}}$	486.0 ± 2.9

The experimental results demonstrate that PASS successfully combines the advantages of semi- and full-gradient training. As shown in Figure 3, PASS benefits from Semi's rapid initial progress and reduces the initial error substantially faster than Full. Switching to full gradients then avoids the persistent oscillations observed when Semi is used throughout training, allowing steadier refinement and the lowest mean final relative L_2 error among the compared methods. The weighted SGR method does not fully recover these benefits. Its semi-gradient contribution also accelerates early progress, but Figure 3 shows that its relative L_2 error largely plateaus after 50,000 steps and remains close to Semi's in late training. This behavior suggests that retaining the semi-gradient contribution throughout training limits later refinement and final solution accuracy.

Fixed switching is also sensitive to the handoff time: the 10%, 25%, and 50% schedules become nonfinite, while Fixed (75%) remains stable but finishes with substantially higher error than PASS.

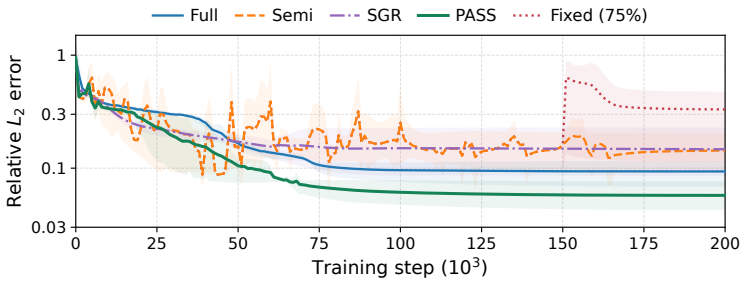

Figure 3: Relative L_2 error during training on Burgers. Fixed (75%) switches from Semi to Full after 75% of the training updates. Lines show means and shaded bands show ranges over three random seeds, without smoothing. PASS combines rapid initial progress with a lower final error than SGR.

As Figure 2 shows, Semi can pass through strongly oscillatory states; we hypothesize that switching at a predetermined time may restore full differentiation from an unfavorable state with poorly scaled PDE derivatives, leading to unstable refinement. By monitoring directional changes in the predictions, PASS can use such oscillatory behavior as a signal to select an appropriate handoff time.

Table 3 further demonstrates PASS's advantages in final accuracy and computational efficiency. Its mean final relative L_2 error is 38.6% lower than Full's, even though both methods use the same full-gradient update rule and learning-rate schedule during later training. We hypothesize that the rapid progress of the initial semi-gradient phase brings the model closer to the solution and provides a more favorable starting point for accurate, numerically stable full-gradient refinement. PASS may therefore enhance the effectiveness of subsequent full-gradient optimization as well as combine the advantages of the two paths. Including detection and switching, PASS has a slightly lower mean total training time than Full (486.0 versus 489.8 s), showing that these operations do not increase the overall training cost in this comparison. These results support both the effectiveness of PASS's switch and the rationale for using the two backward paths at different training stages.

4.3 Reaction–Diffusion Equation

To evaluate the compatibility of PASS with different optimizers, network architectures, and existing PINN training techniques, we conduct experiments on a reaction–diffusion problem of the form studied by Krishnapriyan et al. (2021), with diffusion and reaction coefficients both set to 4:

$$u_t - 4u_{xx} - 4u(1-u) = 0, \qquad (x,t) \in [0, 2\pi] \times [0, 1], \tag{11}$$

with a Gaussian initial profile and periodic value and slope constraints. All comparisons use 50,000 training steps.

Optimizers and architectures. We first evaluate the compatibility of PASS with different network architectures and optimizers. We compare momentum-free SGD and Adam (Kingma & Ba, 2015) on an MLP, then evaluate Adam with a ResNet (He et al., 2016) and a trainable Gaussian Fourier-feature model (Tancik et al., 2020). Within each configuration, Full, Semi, SGR, and PASS share the same initialization, model, optimizer, learning rate, and schedule.

Table 4: Final relative L_2 error ($\times 10^{-3}$) on the reaction–diffusion equation (11) with different optimizers and architectures after 50,000 updates. Results are means $\pm$ sample SD over three random seeds. NaN (2/3) marks two failed runs.

Configuration	Full	Semi	SGR	PASS
MLP / SGD	NaN (2/3)	52.9 ± 22.9	38.7 ± 7.98	$\mathbf{31.7 \pm 3.81}$
MLP / Adam	949 ± 15	2.65 ± 1.06	1.77 ± 2.09	$\mathbf{1.14 \pm 1.06}$
ResNet / Adam	314 ± 540	3.14 ± 1.80	1.52 ± 0.524	$\mathbf{1.26 \pm 0.246}$
Fourier / Adam	295 ± 505	3.57 ± 0.639	0.924 ± 0.290	$\mathbf{0.808 \pm 0.470}$

The SGD results in Table 4 show the benefit of an initial semi-gradient phase followed by full-gradient refinement. Full becomes nonfinite in two of three runs, whereas Semi and PASS complete all three.

PASS reduces the mean final relative L_2 error from Semi's 5.29×10^{-2} to 3.17×10^{-2}. Thus, in this setting, the initial semi-gradient phase enables successful full-gradient refinement and improves accuracy over continued Semi training.

PASS remains effective when momentum-based optimization and different network representations are introduced. With Adam, PASS achieves the lowest mean final error on all three tested architectures (Table 4). Although SGR improves on both fixed paths in these cases, PASS further reduces its mean error by 35.8%, 17.1%, and 12.6% on the MLP, ResNet, and Fourier-feature models, respectively. These improvements show that using the two backward paths at different stages remains beneficial when the optimizer dynamics or network representation changes. Including prediction monitoring and switching, PASS's mean total training time is slightly lower than Full's in all three Adam configurations (Table 12). The accuracy gains therefore do not require an increase in total training cost relative to Full in these comparisons. SGR combines both backward paths at every update, incurring higher training cost than Full in the same three configurations (Table 12).

Training techniques. To further evaluate whether PASS can improve established PINN training methods, we hold MLP and Adam fixed and add one technique at a time: residual-based adaptive sampling (RAD) (Wu et al., 2022), NTK loss reweighting (Wang et al., 2022b), or causal training (Wang et al., 2022a). All comparisons use learning rate 10^{-3} and the same update budget and observation interval as above. Each technique uses the same configuration and random-key schedule across the four methods; adaptive points or weights evolve with the corresponding training trajectory. PASS continues to use the original fixed probe set. Detailed settings are provided in Appendix C.3.

Table 5: Final relative L_2 error ($\times10^{-3}$) on the reaction–diffusion equation (11) with MLP/Adam and different training techniques after 50,000 updates.

Technique	Full	Semi	SGR	PASS
None	949 ± 15	2.65 ± 1.06	1.77 ± 2.09	$\mathbf{1.14 \pm 1.06}$
RAD	595 ± 516	0.958 ± 0.314	0.323 ± 0.0914	$\mathbf{0.119 \pm 0.0410}$
NTK	947 ± 14.7	2.19 ± 0.505	2.25 ± 2.75	$\mathbf{1.09 \pm 1.04}$
Causal	1.47 ± 1.09	2.17 ± 0.812	1.34 ± 0.886	$\mathbf{0.971 \pm 0.490}$

PASS provides additional accuracy gains when combined with each of the three training techniques, achieving the lowest mean final relative L_2 error in every reported configuration (Table 5). The gains are particularly pronounced with RAD, while improvements with NTK reweighting and causal training show that the benefit persists under different loss-weighting schemes. RAD changes where the PDE residual is evaluated, and NTK reweighting changes the relative weights of the loss terms, while PASS changes the residual's backward path. These results show that PASS complements the tested training techniques and can further improve solution accuracy.

With causal training, PASS also reduces the time required to reach a useful accuracy level. The first recorded relative L_2 error below 0.01 is reached in 30.4 s on average, compared with 39.2 s for Full, with an earlier crossing in every run (Appendix C.4). Including monitoring and switching, mean total training time remains slightly lower than Full's (200.5 versus 201.3 s; Table 13). The experiments therefore demonstrate that PASS can complement the tested optimizers, architectures, and training techniques, improving final accuracy while retaining the computational efficiency of full-gradient training.

5 Conclusion

We introduced PASS to combine the numerical benefits of semi-gradients with the refinement capability of full gradients in PINN training. Our computation graph analysis and experiments showed that bypassing parameter differentiation through PDE derivatives can improve stability and reduce per-update cost, while continued semi-gradient training can limit final accuracy. PASS uses changes in network predictions to decide when to restore full-gradient backpropagation. Across the tested configurations, this strategy improves final accuracy and complements existing optimizers, network architectures, and training techniques. These findings support adapting the residual's backward path across training stages to exploit the strengths of both paths.

AI USE STATEMENT

In this work, we used generative AI tools to formulate mathematical claims, provide key elements of mathematical proofs, implement methods, assist with translation. We also used these tools to summarize or analyze existing literature. We did not use generative AI tools to design or provide feedback on research methodology or experiments, develop theoretical models or conceptual frameworks, support qualitative or thematic data analysis, interpret results, generate synthetic datasets, propose or refine hypotheses, or clean and reformat datasets. We have reviewed all AI-assisted work and take responsibility for the final content, including any text, claims, or artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 specifies the two backward paths, the handoff rule, and the conditioning calculation. Appendices A, B, and C provide the PDEs, reference solutions, sampling procedures, and training hyperparameters for the corresponding experiments. They also describe hyperparameter selection and report computation time as mean $\pm$ sample SD.

REFERENCES

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. URL https://web.stanford.edu/~boyd/cvxbook/.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL https://github.com/jax-ml/jax.

Wenbo Cao and Weiwei Zhang. Overcoming the loss conditioning bottleneck in optimization-based PDE solvers: a well-conditioned loss function. *Communications in Nonlinear Science and Numerical Simulation*, 160:109952, 2026. doi: 10.1016/j.cnsns.2026.109952. URL https://arxiv.org/abs/2508.02692.

Junwoo Cho, Seungtae Nam, Hyunmo Yang, Seok-Bae Yun, Youngjoon Hong, and Eunbyung Park. Separable physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/4af827e7d0b7bdae6097d44977e87534-Abstract-Conference.html.

Di He, Shanda Li, Wenlei Shi, Xiaotian Gao, Jia Zhang, Jiang Bian, Liwei Wang, and Tie-Yan Liu. Learning physics-informed neural networks without stacked back-propagation. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pp. 3034–3047, 2023. URL https://proceedings.mlr.press/v206/he23a.html.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2026. URL http://github.com/google/flax.

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3:422–440, 2021. doi: 10.1038/s42254-021-00314-5. URL https://www.nature.com/articles/s42254-021-00314-5.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://papers.nips.cc/paper/2021/hash/df438e5206f31600e6ae4af72f2725f1-Abstract.html.

Xi'an Li, Jinran Wu, Lei Zhang, and Xin Tai. Solving a class of high-order elliptic PDEs using deep neural networks based on its coupled scheme. *Mathematics*, 10(22):4186, 2022. doi: 10.3390/math10224186. URL https://doi.org/10.3390/math10224186.

Lingshi Meng, Haosen Shi, and Sinno Jialin Pan. Deep coupling learning for solving PDEs. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/66309.

Brendan O'Donoghue and Emmanuel Candès. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15(3):715–732, 2015. doi: 10.1007/s10208-013-9150-3. URL https://arxiv.org/abs/1204.3982.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. doi: 10.1016/j.jcp.2018.10.045.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2006.10739.

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

Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022b. doi: 10.1016/j.jcp.2021.110768. URL https://arxiv.org/abs/2007.14527.

Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *arXiv preprint arXiv:2207.10289*, 2022. URL https://arxiv.org/abs/2207.10289.

A BIHARMONIC EQUATION

A.1 PDE AND EVALUATION

For the experiment in Section 4.1, we use the biharmonic test problem considered by Li et al. (2022):

$$\begin{cases} \Delta^2 u = 4\pi^4 \sin(\pi x)\sin(\pi y), & (x,y) \in [0,1]^2, \\ u = \Delta u = 0, & (x,y) \in \partial[0,1]^2. \end{cases} \tag{12}$$

The exact solution is $u^\star(x,y) = \sin(\pi x)\sin(\pi y)$. We parameterize $u_\theta = x(1-x)y(1-y)v_\theta$, where v_θ is a tanh MLP, so that the value boundary condition is satisfied exactly. The remaining loss is

$$L = \mathrm{MSE}\big(\Delta^2 u_\theta - 4\pi^4 \sin(\pi x)\sin(\pi y)\big) + 10\,\mathrm{MSE}_{\partial\Omega}(\Delta u_\theta). \tag{13}$$

Interior points are sampled uniformly over the square; boundary points are divided equally among its four edges and sampled uniformly along each edge. The points remain fixed throughout training. We evaluate relative L_2 error against the analytic solution on a fixed 128×128 grid.

A.2 HYPERPARAMETERS AND COMPARISON PROTOCOL

Table 6 summarizes the training settings. Dense kernels use LeCun-normal initialization and biases are zero. All experiments in this paper use JAX/Flax, float64 arithmetic, and NVIDIA RTX 4090 GPUs. Each setting is repeated with three random seeds; paired methods share their initial parameters and training points. There is no gradient clipping or gradient-norm matching.

Table 6: Training hyperparameters for the biharmonic experiment.

Hyperparameter	Value
Backbone	MLP, four hidden layers, width 64, tanh
Optimizer	SGD without momentum
Training updates	10,000
Interior / boundary points	1,024 / 128
PDE / boundary loss weights	1 / 10
Learning-rate schedule	Constant; no warmup
Error evaluation interval	100 updates
SGR weight ω	0.9
Repeated runs	Three random seeds

We evaluate the same learning-rate sweep for Full, Semi, and SGR: 10^{-5}, 3×10^{-5}, 10^{-4}, 3×10^{-4}, 10^{-3}, and 3×10^{-3}. The sweep is used to compare numerical stability and accuracy, rather than select a separate rate for each method. Table 7 reports all rates, extending the comparisons in Tables 1 and 2. Nonfinite losses, parameters, or optimizer states terminate a run; failed runs are retained when reporting numerical stability.

The SGR baseline is defined as

$$g_{\mathrm{SGR}} = (1-\omega)\frac{2}{n}J_r^\top r_\theta + \frac{\omega}{n}J_u^\top r_\theta + \nabla_\theta B, \tag{14}$$

with $\omega = 0.9$ for all three PDEs. Under our loss normalization, its PDE direction is $0.1g_F^{\mathrm{PDE}} + 0.45g_S^{\mathrm{PDE}}$, and boundary and initial-condition terms retain their ordinary gradients.

A.3 COMPUTATION TIME

Table 8 summarizes cost at the common finite learning rate 10^{-5}. Throughout the paper, total training time includes compilation, synchronized optimizer updates, PASS observations and decisions, and any training-technique computations. It excludes reference evaluation, offline diagnostics, and ordinary checkpoint I/O. Per-update time measures synchronized update execution alone. Methods within each setting use the same physical GPU; SGR was timed in later sessions on that GPU. All timings report mean $\pm$ sample SD over the three runs.

For the accuracy-threshold comparison in Section 4.1, Full at 10^{-5} first reaches relative $L_2 < 0.01$ after 3867 ± 643 updates (32.18 ± 4.42 s), while Semi at 3×10^{-4} reaches it after 1967 ± 764 updates

Table 7: Final relative L_2 error across the complete biharmonic learning-rate sweep. Entries are mean $\pm$ sample SD over three random seeds after 10,000 updates; NaN indicates failure in all three runs.

Learning rate	Full	Semi	SGR
10^{-5}	$(2.62 \pm 0.80) \times 10^{-3}$	$(2.07 \pm 0.10) \times 10^{-1}$	$(2.38 \pm 0.79) \times 10^{-3}$
3×10^{-5}	$(9.74 \pm 16.56) \times 10^{-2}$	$(2.25 \pm 0.62) \times 10^{-2}$	$(3.62 \pm 1.99) \times 10^{-4}$
10^{-4}	$(2.98 \pm 0.25) \times 10^{-1}$	$(4.82 \pm 2.62) \times 10^{-3}$	$(1.15 \pm 0.19) \times 10^{-3}$
3×10^{-4}	NaN	$(1.74 \pm 0.43) \times 10^{-3}$	$(2.89 \pm 4.55) \times 10^{-2}$
10^{-3}	NaN	$(3.64 \pm 2.53) \times 10^{-3}$	$(1.21 \pm 0.37) \times 10^{-1}$
3×10^{-3}	NaN	$(6.60 \pm 2.18) \times 10^{-3}$	NaN

Table 8: Biharmonic computation time at learning rate 10^{-5}. Total time covers 10,000 updates; per-update time excludes compilation. Entries are mean $\pm$ sample SD over three random seeds.

Measurement	Full	Semi	SGR
Total training (s)	73.47 ± 0.07	31.35 ± 0.10	74.36 ± 0.02
Per update (ms)	6.73 ± 0.01	2.86 ± 0.00	6.83 ± 0.00

(8.26 ± 2.26 s). These are the rates giving each method its lowest final error in Table 2. Threshold times use the first recorded evaluation, without interpolation; all runs continue to the full training budget.

B BURGERS EQUATION

B.1 PDE AND EVALUATION

For Section 4.2, we use the viscous Burgers benchmark considered by Raissi et al. (2019):

$$\begin{cases} u_t + uu_x - \dfrac{0.01}{\pi} u_{xx} = 0, \quad (x,t) \in [-1,1] \times [0,1], \\ u(x,0) = -\sin(\pi x), \\ u(-1,t) = u(1,t) = 0. \end{cases} \tag{15}$$

We sample fixed, uniformly random interior, boundary, and initial-condition points. The loss assigns weight 1 to the PDE residual and weight 10 to each constraint term. No hard boundary envelope is used. Relative error is evaluated against the supplied numerical reference in `burgers.mat`, using its full 201×512 time–space grid without interpolation. Reference values are used only for evaluation, not for gradient updates or online switching.

B.2 HYPERPARAMETERS AND SWITCHING

The MLP uses the same hidden-layer dimensions, activation, and initialization as the biharmonic backbone, with a scalar linear output. Table 9 gives the remaining settings. The same optimizer, learning rate, and training budget are used for Full, Semi, SGR, and PASS.

Table 9: Training hyperparameters for the Burgers experiment.

Hyperparameter	Value
Backbone	MLP, four hidden layers, width 64, tanh
Optimizer	SGD without momentum
Training updates	200,000
Interior / boundary / initial points	1,024 / 256 / 256
PDE / boundary / initial loss weights	1 / 10 / 10
Base learning rate	3×10^{-3}
Warmup / decay	2,000 updates / cosine
Minimum learning-rate multiplier	0.01
PASS observation interval K	500 updates
Error evaluation interval	1,000 updates
SGR weight ω	0.9
Repeated runs	Three random seeds

For a budget T, the learning rate at zero-indexed update s is $\eta_s = \eta_0 m(s)$, where

$$m(s) = \begin{cases} (s+1)/2000, & 0 \le s < 2000, \\ 0.01 + 0.99\{1 + \cos[\pi(s-2000)/(T-2000)]\}/2, & 2000 \le s < T. \end{cases} \tag{16}$$

PASS uses the original interior points as its fixed prediction probe. Observations start at step zero, so the first direction comparison is available after $2K$ updates. The switch preserves the optimizer state and global learning-rate counter; no decision is made at the final training step. A nonpositive product of displacement norms is assigned zero cosine, while a missing angle leaves the path unchanged. A nonfinite angle passed to the policy selects Full and records a failure flag; nonfinite training states terminate the run.

B.3 HYPERPARAMETER SELECTION AND TIMING

The candidate base learning rates were 10^{-4}, 3×10^{-4}, 10^{-3}, 3×10^{-3}, and 10^{-2}. We selected 3×10^{-3} by minimizing Full's final relative L_2 error on a separate development run for each candidate rate. All methods use this same rate, so the comparison is based on the strongest Full setting found in this search.

C REACTION–DIFFUSION EQUATION

C.1 PDE AND REFERENCE SOLUTION

The experiment in Section 4.3 uses a reaction–diffusion problem of the form studied by Krishnapriyan et al. (2021), with diffusion and reaction coefficients both set to 4:

$$\begin{cases} u_t - 4u_{xx} - 4u(1-u) = 0, & (x,t) \in [0, 2\pi] \times [0,1], \\ u(x,0) = \exp\left[-\frac{1}{2}\left(\frac{x-\pi}{\pi/4}\right)^2\right], & \\ u(0,t) = u(2\pi,t), \qquad u_x(0,t) = u_x(2\pi,t). & \end{cases} \tag{17}$$

The loss is

$$L = \lambda_r \,\mathrm{MSE}(r) + \lambda_b\{\mathrm{MSE}(b_u) + \mathrm{MSE}(b_{u_x})\} + \lambda_i \,\mathrm{MSE}(b_{\mathrm{IC}}), \tag{18}$$

where b_u and b_{u_x} are the periodic value and slope mismatches. The initial weights are $(\lambda_r, \lambda_b, \lambda_i) = (1, 10, 10)$; both boundary terms retain ordinary gradients under all methods. Interior and initial points are sampled uniformly, and each boundary pair shares a uniformly sampled time at $x = 0$ and $x = 2\pi$. Points remain fixed except when RAD refreshes the interior set.

We compute the numerical reference by Strang splitting, using an exact logistic reaction half-step, an FFT diffusion step, and a second reaction half-step. The internal time step is 10^{-4} on 512 uniformly spaced periodic spatial points, excluding the duplicate endpoint. We save 201 equally spaced times and evaluate relative error on the complete time–space grid. The same reference is used for all methods and never enters training or online path decisions.

C.2 ARCHITECTURES AND HYPERPARAMETERS

Table 10 gives the settings shared by the reaction–diffusion comparisons. The MLP has four width-64 tanh layers. The ResNet (He et al., 2016) uses the same backbone, replacing each hidden transition after the first by $(h + \tanh(Wh + b))/\sqrt{2}$. The Fourier-feature model (Tancik et al., 2020) maps raw coordinates $z = [x, t]$ to $[\cos(zB), \sin(zB)]$ before the MLP. Here $B \in \mathbb{R}^{2\times 128}$ is initialized with independent standard Gaussian entries and trained jointly with the network. Raw coordinates are not appended, and no π factor is inserted. All dense layers use LeCun-normal kernels and zero biases.

Table 10: Shared reaction–diffusion training hyperparameters. Architecture-specific learning rates are given in Table 11.

Hyperparameter	Value
Hidden layers / width / activation	4 / 64 / tanh
Training updates	50,000
Interior / boundary pairs / initial points	1,024 / 256 / 256
Initial PDE / boundary / initial weights	1 / 10 / 10
SGD momentum	0
Adam $(\beta_1, \beta_2, \epsilon)$	$(0.9, 0.999, 10^{-8})$
Warmup / decay	2,000 updates / cosine
Minimum learning-rate multiplier	0.01
PASS observation interval K	1,000 updates
Error evaluation interval	1,000 updates
SGR weight ω	0.9
Repeated runs	Three random seeds

Adam follows Kingma & Ba (2015). We use the schedule in Eq. (16) with $T = 50{,}000$ and the PASS observation and state-preservation rules in Appendix B.2. The prediction probe is always the initial interior set, including in the RAD experiment. Within each configuration, all methods share the initial parameters, optimizer state, and point sets.

Table 11: Learning-rate selection for reaction–diffusion. All methods within a configuration use the same selected rate. The Fourier rate is inherited from earlier development.

Configuration	Candidate rates	Selected rate
MLP / SGD	$10^{-4}, 10^{-3}, 10^{-2}$	10^{-3}
MLP / Adam	$3\times10^{-4}, 10^{-3}, 3\times10^{-3}$	10^{-3}
ResNet / Adam	$3\times10^{-4}, 10^{-3}, 3\times10^{-3}$	10^{-3}
Fourier / Adam	No additional search	3×10^{-4}

C.3 Training techniques

We add RAD, NTK reweighting, and causal training separately to the same MLP/Adam configuration. Technique settings and random-key schedules are shared across methods; adaptive points and weights depend on each model's own trajectory.

Residual-based adaptive sampling. Following RAD (Wu et al., 2022), every 1,000 updates, starting at zero, we sample 8,192 uniformly distributed candidate points and evaluate their physical residual magnitudes $a_i = |r_i|$. Sampling probabilities are proportional to $a_i/\operatorname{mean}(a) + 1$, with uniform probabilities if all residuals vanish. We draw 1,024 points without replacement and hold them fixed until the next refresh. The residual loss has no importance-weight correction; boundary and initial-condition points remain unchanged.

NTK loss reweighting. Every 1,000 updates, starting at zero, we use 128 points for each of the PDE, boundary-pair, and initial-condition sets. From the full physical parameter Jacobians, irrespective of the active backward path, we compute

$$d_r = \operatorname{mean}\|\nabla_\theta r_i\|^2, \qquad d_i = \operatorname{mean}\|\nabla_\theta b_{\mathrm{IC},i}\|^2, \tag{19}$$

$$d_b = \operatorname{mean}\|\nabla_\theta b_{u,i}\|^2 + \operatorname{mean}\|\nabla_\theta b_{u_x,i}\|^2. \tag{20}$$

With $\bar{d} = (d_r + d_b + d_i)/3$, the target for each loss weight is $t_j = \bar{d}/\max(d_j, \mathrm{tiny})$, where tiny is the smallest positive normalized float64 value. We update $\lambda_j \leftarrow 0.9\lambda_j + 0.1t_j$, starting from $(1, 10, 10)$ without an additional multiplication by these initial weights. If all statistics vanish, the targets are one. Targets and weights are detached and not clipped. This uses finite-probe NTK scaling (Wang et al., 2022b).

Causal training. We sort the 1,024 interior points by time and divide them into 32 equally sized blocks. For the physical mean squared residual E_j in block j, the PDE loss is

$$L_r = \frac{1}{32}\sum_{j=1}^{32} w_j E_j, \qquad w_j = \operatorname{stopgrad}\left[\exp\left(-\sum_{\ell<j} E_\ell\right)\right]. \tag{21}$$

The causal tolerance is one. Weights are recomputed at every update, without normalization or boundary/initial losses in the exponent, following the causal weighting principle (Wang et al., 2022a). Full and semi use identical forward weights at the same parameters.

C.4 Computation time

Tables 12 and 13 report total training time using the definition in Appendix A.3. They include monitoring, switching, and training-technique computations, and summarize each configuration by mean $\pm$ sample SD. For RAD, time spent saving the sampling record is included in its technique cost. Failed runs are timed to termination and are not interpreted as completed-budget speedups.

In the causal comparison, the first recorded relative $L_2 < 0.01$ is reached after 30.38 ± 16.59 s with PASS and 39.19 ± 20.07 s with Full. These are means $\pm$ sample SD across the same three runs. Thresholds are evaluated every 1,000 updates without interpolation; all runs continue to 50,000 updates.

Table 12: Reaction–diffusion training time (s) across optimizers and architectures, mean ± sample SD over three random seeds. The budget is 50,000 updates. † includes two failed Full runs timed to termination.

Configuration	Full	Semi	SGR	PASS
MLP / SGD	$68.38 \pm 108.69^{\dagger}$	137.78 ± 0.01	204.48 ± 0.07	192.76 ± 0.04
MLP / Adam	203.55 ± 0.11	146.57 ± 0.02	215.57 ± 0.08	200.35 ± 3.84
ResNet / Adam	204.90 ± 0.08	150.49 ± 0.02	218.54 ± 0.58	201.46 ± 4.94
Fourier / Adam	297.53 ± 0.56	193.91 ± 0.19	312.69 ± 0.33	294.41 ± 1.17

Table 13: Reaction–diffusion training time (s) with MLP/Adam and different training techniques, mean ± sample SD over three random seeds after 50,000 updates. The None row uses a separate timing repeat of the MLP/Adam configuration.

Technique	Full	Semi	SGR	PASS
None	203.61 ± 0.13	146.18 ± 0.09	215.66 ± 0.08	200.14 ± 3.92
RAD	201.25 ± 0.57	144.89 ± 0.40	212.57 ± 0.17	198.77 ± 1.08
NTK	211.19 ± 0.00	153.70 ± 0.07	223.25 ± 0.23	209.37 ± 1.41
Causal	201.32 ± 0.16	145.29 ± 0.07	231.72 ± 0.42	200.51 ± 0.04