LEARNING PHYSICS-CONSTRAINED COLLOCATION SAMPLERS FOR DUAL-CONE OPTIMIZATION IN PINNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Physics-informed neural networks (PINNs) are highly sensitive to the choice of collocation points used to enforce PDE residuals and boundary conditions, particularly when trained with modern optimization methods such as dual cone gradient descent that rely on stable gradient geometry. Existing sampling strategies are largely heuristic and fail to adapt to the optimization dynamics induced by these optimizers, leading to poor sample efficiency and unstable training in high-dimensional and complex PDEs. We propose CoDyS (Collocation for Dynamic Sampling), a framework that learns physics-constrained collocation samplers to optimize PINN training under a fixed dual cone gradient descent optimizer. CoDyS formulates sampler design as a bi-level learning problem in which a generator agent proposes structured, physics-valid sampling programs, a PINN agent executes training with a fixed architecture and optimizer, and a critic agent evaluates candidate samplers using held-out PDE residual and boundary losses together with dual-cone stability diagnostics. Final performance is then reported with an independent relative L^2 error metric on a disjoint test set. Without modifying the PINN or optimizer, CoDyS consistently improves sample efficiency and training stability compared to space-filling and adaptive sampling baselines across benchmark and high-dimensional PDEs, while remaining compatible with consumer-grade local workstations for open-weight models and lightweight API access for proprietary generators, without requiring model fine-tuning.

1 INTRODUCTION

We introduce **CoDyS**, a framework for learning physics-constrained collocation samplers that optimize the training dynamics of physics-informed neural networks under geometry-aware optimization. Physics-informed neural networks (PINNs) provide a flexible, mesh-free approach for solving partial differential equations (PDEs) by embedding physical constraints directly into the training objective through PDE residual and boundary condition losses Raissi et al. (2019). This formulation has enabled PINNs to address a wide range of scientific computing problems, including nonlinear, high-dimensional, and parametric PDEs Lu et al. (2021); Pang & Karniadakis (2019). Despite their expressive power, PINNs frequently exhibit slow convergence, instability, or convergence to physically invalid solutions, particularly as problem dimension and stiffness increase Krishnapriyan et al. (2021); Wang et al. (2021).

A growing body of work has identified optimization as a central bottleneck in PINN training. Prior approaches have proposed adaptive loss balancing Wang et al. (2022b); Zhou et al. (2025), gradient manipulation Yu et al. (2020), and multi-objective optimization strategies Liu et al. (2021) to mitigate conflicts between PDE residual and boundary losses. More recently, dual cone gradient descent (DCGD) has provided a principled geometric interpretation of PINN optimization by constraining parameter updates to a dual cone region in which both loss components decrease simultaneously Hwang & Lim (2024). By explicitly reasoning about gradient angles and magnitude ratios, DCGD improves training stability and convergence and establishes a theoretical connection between PINN training and multi-objective optimization.

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.

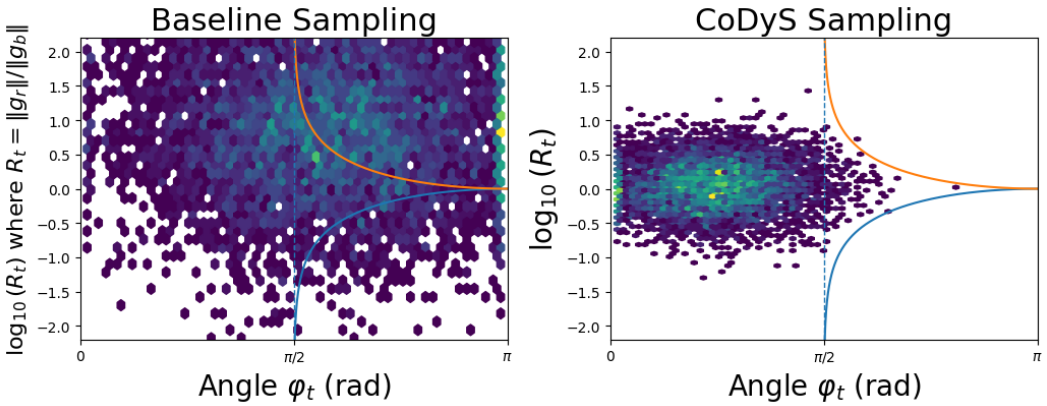

Figure 1: Learned collocation sampling behavior for Helmholtz PDE. The CoDyS policy converges to a balanced interior–boundary sampling regime with moderate exploration, corresponding to stable gradient geometry.

Collocation sampling is itself a mature line of PINN research rather than an unaddressed design choice. Prior work includes grids, random and low-discrepancy space-filling rules; residual-based adaptive refinement and redistribution; loss-proportional importance sampling; retain–resample–release policies; causal weighting for temporal PDEs; and curriculum-based emphasis on difficult points Wu et al. (2023); Nabian et al. (2021); Daw et al. (2023); Wang et al. (2022a); Duan et al. (2025). These methods provide strong, inexpensive inductive biases, and several improve substantially over uniform sampling in their target regimes. Most instantiate a predefined acquisition statistic or update rule, however, so choosing among them and coordinating interior and boundary sampling remain problem-dependent. The gap studied here is therefore not whether adaptive sampling helps, but whether the sampler can be treated as an executable, compositional policy that is selected using the behavior of a fixed inner optimizer over an entire training trajectory.

We address this limitation by treating the sampled objective itself as an optimization variable. CoDyS is a bi-level framework that optimizes the operating conditions of DCGD: the inner loop keeps the network, loss definition, and DCGD update rule fixed, while the outer loop searches for a sampling policy that improves the trajectory induced by that optimizer. Since collocation points determine the finite-sample residual and boundary objectives, changing their distribution changes the local loss values, gradients, gradient variance, and curvature encountered by DCGD. CoDyS is therefore more than a choice among standard samplers: it is an optimizer-conditioned program search that shapes the PDE-specific empirical loss landscape on which DCGD acts. Thus, our evaluation answers two complementary questions. First, comparison with optimizer-centric baselines tests whether changing the sampled objective can add value beyond transforming gradients produced by a fixed sampler. Second, the direct sampler comparison in Table 2 holds DCGD, the network, loss, training horizon, and validation protocol fixed while comparing CoDyS with a default fixed sampler, a hand-designed interior–boundary allocation rule, and a residual-adaptive sampler. This comparison directly tests whether the gain comes from sampling, whether adaptivity alone explains it, and whether trajectory feedback contributes beyond a fixed acquisition rule.

2 Preliminaries

Notation. Let $\Omega \subset \mathbb{R}^d$ denote a bounded domain with boundary $\partial\Omega$. We consider partial differential equations of the form

$$\mathcal{N}[u](x) = f(x), \qquad x \in \Omega, \tag{1}$$

with boundary conditions

$$\mathcal{B}[u](x) = g(x), \qquad x \in \partial\Omega. \tag{2}$$

A neural network $u(x;\theta)$ with parameters θ is used to represent the unknown solution. We denote by $\mathcal{X}_r = \{x_r^i\}_{i=1}^{N_r} \subset \Omega$ the set of interior collocation points and by $\mathcal{X}_b = \{x_b^i\}_{i=1}^{N_b} \subset \partial\Omega$ the set of boundary collocation points. The residual and boundary loss functions are written as $\mathcal{L}_r(\theta)$ and $\mathcal{L}_b(\theta)$, with corresponding gradients $g_r = \nabla_\theta \mathcal{L}_r$ and $g_b = \nabla_\theta \mathcal{L}_b$. We use $\langle \cdot, \cdot \rangle$ to denote the

Euclidean inner product. Throughout the paper, collocation sets are generated by a sampling policy π, and network parameters are updated using a fixed optimizer.

Related Works. Physics-informed neural networks (PINNs) (Raissi et al., 2019) solve PDEs with residual and boundary losses, but training often suffers from gradient imbalance and objective conflict (Krishnapriyan et al., 2021; Wang et al., 2021). Prior work mainly addresses this through optimization. LRA adapts step sizes from gradient statistics (Wang et al., 2021), NTK methods rebalance losses spectrally (Wang et al., 2022b), and dual-balancing reweights objectives by training difficulty (Zhou et al., 2025). PCGrad (Yu et al., 2020), CAGrad (Liu et al., 2021), and AGDA (Li et al., 2023) modify gradients to reduce conflict, while ENGD (Muller & Zeinhofer, 2023), ANaGRAM (Schwencke & Furtlehner, 2025), and randomized Gram-sketching variants (McKay et al., 2025) use function-space geometry. Most closely related, DCGD (Hwang & Lim, 2024) constrains updates to a dual-cone region to mitigate conflict. Complementary sampling work spans space-filling and residual-adaptive designs (Wu et al., 2023), importance sampling (Nabian et al., 2021), residual retention (Daw et al., 2023), causal training (Wang et al., 2022a), and curricula (Duan et al., 2025); each encodes a different coverage or refinement bias.

Novelty relative to adaptive sampling. CoDyS does not claim to introduce adaptive collocation sampling. Its distinction is the search object and feedback loop: instead of prescribing one acquisition statistic, it searches executable programs that can compose point generators, density transforms, and joint interior–boundary allocation rules. Candidate programs are ranked by repeat-averaged reference-based validation error after a complete fixed-optimizer rollout; residual, loss-balance, and optimizer-geometry summaries inform subsequent proposals but are not added to the ranking objective. Agents of Diffusion (Khanal et al., 2026) motivates generator–critic control of data distributions, whereas CoDyS applies this pattern to physics-constrained sampler programs and optimizer-conditioned PINN trajectories. PINNacle (Hao et al., 2024) provides large-scale PDE benchmarking, and PINNsAgent (Wuwu et al., 2025) uses multi-agent LLMs to optimize PINN architectures; neither is presented here as a sampler baseline.

General PINNs can contain more than two independently conflicting losses. ConFIG constructs an update having positive inner product with every loss-specific gradient (Liu et al., 2025), while HARMONIC seeks directions in the intersection of primal and dual gradient cones to obtain feasible, non-conflicting multi-loss updates (Bu et al., 2026). These methods address a broader loss geometry than the two-vector DCGD instantiation studied here and motivate the multi-loss extension.

Physics-Informed Neural Networks. Physics-informed neural networks (PINNs) solve PDEs by optimizing the parameters θ of the neural approximation $u(x;\theta)$ so that it satisfies both the governing equation and the boundary conditions at the collocation points defined above. Training is formulated as the minimization of a weighted composite objective

$$\min_{\theta} \; \mathcal{L}(\theta) \;=\; \lambda_r \mathcal{L}_r(\theta) + \lambda_b \mathcal{L}_b(\theta), \qquad \lambda_r, \lambda_b > 0, \tag{3}$$

where the residual loss enforces the PDE in the interior,

$$\mathcal{L}_r(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \mathcal{N}[u(\cdot;\theta)](x_r^i) - f(x_r^i) \right|^2, \tag{4}$$

and the boundary loss enforces the boundary conditions,

$$\mathcal{L}_b(\theta) = \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \mathcal{B}[u(\cdot;\theta)](x_b^i) - g(x_b^i) \right|^2. \tag{5}$$

Unless stated otherwise, we set $\lambda_r = \lambda_b = 1$ throughout.

This objective induces a multi-objective optimization problem in which distinct physical constraints are enforced through separate loss terms, whose gradients may differ substantially in magnitude and direction. Consequently, the behavior of PINN training depends critically on both the optimization algorithm and the collocation point distributions used to approximate these losses.

The sampler defines the empirical loss landscape. At iteration t, a policy π generates collocation sets $S_t = (\mathcal{X}_r^t, \mathcal{X}_b^t)$ and therefore instantiates the empirical objective

$$\widehat{\mathcal{L}}_{\pi,t}(\theta) = \lambda_r \widehat{\mathcal{L}}_r(\theta; \mathcal{X}_r^t) + \lambda_b \widehat{\mathcal{L}}_b(\theta; \mathcal{X}_b^t). \tag{6}$$

Thus, two samplers applied to the same PDE and network expose the optimizer to different finite-sample landscapes. They can induce different local slopes, curvature, stochastic variation, and relative emphasis between physical constraints even though the underlying continuum problem is unchanged. Throughout the paper, *shaping the loss landscape* refers to controlling this trajectory-local empirical objective and its gradient field; it is not a claim that CoDyS changes the continuum PDE objective or guarantees global smoothness.

Dual Cone Gradient Descent. Dual Cone Gradient Descent (DCGD) constructs an update from the empirical residual and constraint gradients such that both objectives admit a common descent direction (Hwang & Lim, 2024). DCGD determines how to update the network after these gradients are observed; it does not determine which collocation points produce them. CoDyS acts at this upstream stage: it learns the collocation policy while leaving the DCGD update fixed. Our contribution is therefore an optimizer-conditioned sampler that changes the empirical gradient pairs presented to DCGD, not a modification of DCGD itself.

We follow the two-objective DCGD protocol. For non-residual constraints $\{\mathcal{L}_{c,j}\}_{j=1}^m$, we form

$$\mathcal{L}_c = \sum_{j=1}^m \lambda_j \mathcal{L}_{c,j}, \qquad g_c = \nabla_\theta \mathcal{L}_c = \sum_{j=1}^m \lambda_j \nabla_\theta \mathcal{L}_{c,j}, \tag{7}$$

and apply DCGD to (g_r, g_c). CoDyS may sample individual constraints separately, but their gradients are aggregated before the update and geometry diagnostics. The experiments therefore evaluate sampling under this two-objective protocol; they do not establish resolution of conflicts among multiple independently preserved constraints. Such an extension would retain each constraint gradient and pair CoDyS with a multi-loss rule such as ConFIG or HARMONIC.

Terminology. We use *gradient geometry* to denote the per-iteration relationship between the empirical residual and boundary gradients, especially the norm ratio $R_t = \|g_r^t\|/\|g_b^t\|$ and alignment $\cos\varphi_t = \langle g_r^t, g_b^t\rangle/(\|g_r^t\|\,\|g_b^t\|)$. *Optimization-aware feedback* refers to trajectory-level diagnostic summaries built from these quantities and loss-change statistics; these diagnostics guide proposals but are not terms in the ranking objective. We use *bi-level optimization* for the setting in which the inner loop updates θ with a fixed optimizer on sampled collocation points, while the outer loop selects the sampling policy that shapes the resulting training trajectory.

3 Collocation Sampling and PINN Optimization

What is established by prior work. Two findings motivate our study. First, PINNs can exhibit ill-conditioned loss landscapes and strongly imbalanced gradients across physics constraints, which can impede optimization even when the network is sufficiently expressive (Wang et al., 2021; Krishnapriyan et al., 2021). Second, collocation placement materially affects PINN accuracy: broad comparisons include both non-adaptive space-filling rules and residual-adaptive methods, while importance, causal, and retain–resample–release strategies improve particular PDE regimes through different mechanisms (Wu et al., 2023; Nabian et al., 2021; Wang et al., 2022a; Daw et al., 2023). These results establish that both optimization and sampling matter; they do not establish that one sampling family is uniformly superior or that sampling is the sole cause of PINN failure.

A direct consequence of the empirical objective. For fixed parameters θ_t, the sampled sets $(\mathcal{X}_r^t, \mathcal{X}_b^t)$ determine the empirical gradients $g_r^t = \nabla_\theta \widehat{\mathcal{L}}_r(\theta_t; \mathcal{X}_r^t)$ and $g_b^t = \nabla_\theta \widehat{\mathcal{L}}_b(\theta_t; \mathcal{X}_b^t)$. Consequently, the DCGD inputs

$$R_t = \frac{\|g_r^t\|}{\|g_b^t\|}, \qquad \cos\varphi_t = \frac{\langle g_r^t, g_b^t\rangle}{\|g_r^t\|\,\|g_b^t\|} \tag{8}$$

depend on both the PDE and the sampled finite-sample objective. This observation establishes a coupling between sampling and DCGD; by itself, it does *not* imply that a particular sampler causes instability, that reducing dual-cone violations must reduce solution error, or that CoDyS smooths the global continuum loss landscape.

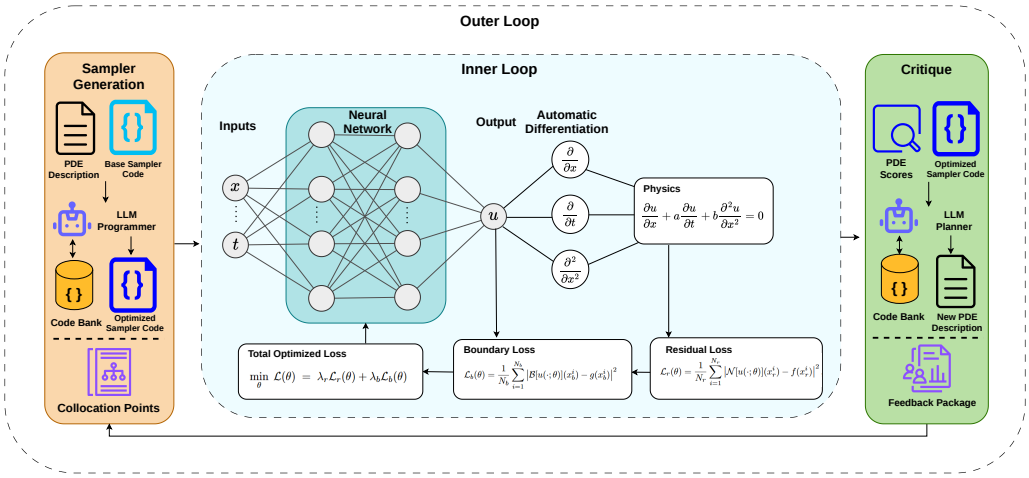

Figure 2: Overview of the CoDyS framework. CoDyS treats collocation sampling as a controllable variable and searches over sampler programs using repeat-averaged reference-based validation error; physics-loss and gradient-geometry quantities are diagnostic feedback.

Working hypothesis and empirical scope. Our hypothesis is narrower: because DCGD acts on sample-dependent gradient pairs, an optimizer-conditioned sampler can sometimes improve the trajectory presented to DCGD under a finite collocation budget. Fixed and quasi-random samplers remain strong, inexpensive choices when broad coverage is sufficient, whereas residual-, importance-, and causal-adaptive methods can be advantageous when errors are localized or temporal propagation is difficult (Wu et al., 2023; Nabian et al., 2021; Daw et al., 2023). We therefore treat these approaches as substantive baselines, not intrinsically deficient heuristics. Figure 1 is an illustration of PDE-dependent sampling behavior which satifies the dual cone behaviour.

Coverage, weighting, and finite-sample effects. Sampling involves a bias–variance and coverage trade-off rather than an unavoidable "sampling bias." Uniform Monte Carlo sampling gives an unbiased estimator of a uniformly weighted continuum residual loss, but a finite realization can have high variance or miss localized structure. Deterministic space-filling rules improve coverage, while adaptive sampling reallocates points toward selected regions and may change the effective weighting of the empirical objective unless importance corrections are used. CoDyS searches this trade-off using repeat-averaged reference-based validation error, with physics-loss and gradient-geometry statistics supplied only as diagnostic feedback. It is therefore a controlled black-box heuristic with validation safeguards, not a theorem that the selected sampler is unbiased, globally optimal, or reliable outside the tested PDE distribution.

4 Methodology

This section develops the paper's central contribution: constrained black-box search over executable collocation programs (Figure 2 and Algorithm 1). The program representation defines what may be changed; the inner loop measures the outcome under a fixed optimizer; and the generator–critic loop proposes subsequent program edits. We first state the minimum formulation needed to expose the sampling–optimizer interface and then describe these three components. In this paper, the inner optimizer is DCGD, so CoDyS is a sampling complement to DCGD rather than a replacement.

4.1 Problem Formulation and Sampling as Control

We consider a physics-informed neural network (PINN) with parameters $\theta \in \mathbb{R}^p$ trained to approximate the solution of a fixed partial differential equation posed on a domain $\Omega \subset \mathbb{R}^d$. The PDE operator $\mathcal{N}$, boundary operator $\mathcal{B}$, network architecture, loss formulation, and optimizer $\mathcal{A}$ are all fixed throughout this work. Training minimizes the composite objective

$$\mathcal{L}(\theta) = \mathcal{L}_r(\theta) + \mathcal{L}_b(\theta), \tag{9}$$

where $\mathcal{L}_r$ and $\mathcal{L}_b$ denote the PDE residual and boundary losses, respectively.

Sampling–optimizer interface. Let π be a probability distribution over finite interior and boundary collocation sets $(\mathcal{X}_r, \mathcal{X}_b)$, with $\mathcal{X}_r \subset \Omega$ and $\mathcal{X}_b \subset \partial\Omega$. PINN training under π follows

$$\theta_{t+1} = \mathcal{A}\big(\theta_t;\, \mathcal{X}_r^t, \mathcal{X}_b^t\big)\,, \qquad (\mathcal{X}_r^t, \mathcal{X}_b^t) \sim \pi, \tag{10}$$

where $\mathcal{A}$ computes empirical losses on the sampled sets and applies the optimizer. Even when $\mathcal{A}$ is deterministic, the realized training trajectory depends on π through the data used to evaluate the loss.

Under this formulation, the residual and boundary gradients at iteration t are $g_r = \nabla_\theta \mathcal{L}_r(\theta_t)$ and $g_b = \nabla_\theta \mathcal{L}_b(\theta_t)$, both empirical estimators induced by the sampled collocation sets $(\mathcal{X}_r^t, \mathcal{X}_b^t)$. Thus, the optimizer operates on gradient pairs whose magnitude, alignment, and variance are controlled by the sampling policy π. This dependence is critical for geometry-aware optimizers such as dual-cone gradient descent, which require either (i) non-conflicting gradients $\langle g_r, g_b\rangle \geq 0$, or (ii) in the conflicting case, a magnitude ratio satisfying $-\cos\varphi_t \leq R_t \leq -1/\cos\varphi_t$, where φ_t is the angle between g_r and g_b Hwang & Lim (2024).

At a fixed iterate θ_t, the norm ratio $R_t = \|g_r\|/\|g_b\|$ and alignment $\cos\varphi_t = \langle g_r, g_b\rangle/(\|g_r\|\|g_b\|)$ are therefore sample-dependent quantities. This elementary observation motivates exposing their trajectory summaries to the proposal mechanism; it is not presented as a theorem and does not imply faster convergence, improved stability, or lower final error for any policy.

Novelty gap in the formulation. Existing adaptive methods can be represented as restricted policy classes Π_s in which a designer fixes an acquisition signal s, such as residual magnitude, a loss-proportional density, retained high-residual points, or temporal causality, and tunes only the associated schedule or threshold. These restrictions are valuable inductive biases and often reduce search cost (Wu et al., 2023; Nabian et al., 2021; Daw et al., 2023; Wang et al., 2022a). CoDyS instead searches a broader program class Π_{prog} whose members may compose generators, spatial transforms, resampling schedules, and separate interior–boundary allocations. Its claimed advantage is flexibility under a fixed optimizer, not a universally better acquisition principle. The empirical question is therefore whether search over Π_{prog} improves on representative fixed and residual-adaptive policies when the inner training procedure is held constant.

$$\pi^* = \operatorname*{arg\,min}_{\pi \in \Pi_{\mathrm{prog}}} \mathbb{E}_{(\mathcal{X}_r^t, \mathcal{X}_b^t)\sim\pi}\left[\mathcal{L}(\theta_T)\right] \quad \text{s.t.} \quad \theta_{t+1} = \mathcal{A}\big(\theta_t; \mathcal{X}_r^t, \mathcal{X}_b^t\big)\,, \quad t = 0, \ldots, T-1. \tag{11}$$

This idealized bi-level problem motivates the search task. In the implemented outer loop, candidate sampler programs are instead compared using the empirical repeat-averaged score $\mathcal{J}(\pi)$.

4.2 Bi-Level Sampler Learning Framework

The analysis in Section 4.1 shows that collocation sampling changes the distribution of gradient geometries encountered during PINN training. Since these geometries influence the behavior of geometry-aware optimizers, sampling is a natural quantity to tune rather than a purely secondary design choice. However, sampling decisions affect training only indirectly, through the resulting optimization trajectory. This motivates a bi-level formulation in which candidate sampler programs are compared through the training dynamics they induce.

Inner Loop (PINN Training). For a fixed sampling policy π, the inner loop follows standard PINN training: at each iteration t, collocation sets $(\mathcal{X}_r^t, \mathcal{X}_b^t) \sim \pi$ are sampled and the parameters are updated by a fixed optimizer $\mathcal{A}$, producing a trajectory $\{\theta_t\}_{t=0}^T$. In our main setting, $\mathcal{A} = \mathrm{DCGD}$. With the optimizer, loss, and architecture fixed, sampling is the only external input shaping the measured training geometry.

Outer Loop (Black-Box Sampler Search). The outer loop updates the current sampler program from trajectory-level feedback. The generator proposes local edits, the PINN executes rollouts under the fixed optimizer, and the critic summarizes diagnostics for the next proposal. The scalar ranking value is computed separately as repeat-averaged reference-based validation error. Since generator and critic weights are fixed, this outer loop is a heuristic black-box program search rather than gradient-based reinforcement learning over generator parameters.

Policy Representation and Update Rule. A sampling policy π is an executable sampler program that maps PDE metadata and a random seed to interior and boundary collocation sets $(\mathcal{X}_r, \mathcal{X}_b)$. Programs combine point generators, allocation rules, simple density transforms, and lightweight control flow, subject to validation of point budgets, domain support, interface correctness, and runtime. In practice, these routines choose a base generator, allocate samples across interior and boundary regions, and apply simple local transforms such as corner emphasis. CoDyS therefore searches over compositional sampler programs rather than selecting a single fixed sampler.

At outer iteration k, the frozen generator proposes a local edit to the current sampler program using the critic summary. A validation wrapper rejects programs that violate the sampler interface, domain support, point budget, or runtime constraints. Each feasible program is then evaluated by repeated PINN training on a fixed seed schedule. The implemented scalar ranking objective is the mean relative L^2 error on a fixed reference-based validation grid; lower is better. PDE residual and boundary losses, gradient-balance statistics, and dual-cone indicators are recorded as diagnostics and returned to the critic, but they are not added to the scalar ranking objective. Intermediate proposals may regress and still seed the next local edit; the procedure separately tracks the lowest-error feasible program observed over the search.

4.3 Optimization-Aware Feedback Signals and Learning Procedure

Given a training trajectory $\{\theta_t\}_{t=0}^{T}$ generated under sampling policy π, we record the norm ratio $R_t = \|g_r^t\|/\|g_b^t\|$, alignment $\cos\varphi_t = \langle g_r^t, g_b^t\rangle/(\|g_r^t\|\|g_b^t\|)$, loss progress, and residual and boundary losses. The critic summarizes these quantities to help the generator propose the next local edit. They do not define or modify the ranking score: candidate and best-so-far comparisons use $\mathcal{J}_{\mathrm{val}}$ alone. The outer loop evaluates sampler programs through inner-loop PINN training. Generator and critic weights remain fixed, so learning operates at the level of programs rather than through parameter updates. The current feasible program is locally edited from one iteration to the next, including after a regressing proposal, while a separate archive retains the feasible program with the lowest $\mathcal{J}_{\mathrm{val}}$ observed so far.

Algorithm 1 CoDyS: Black-Box Outer-Loop Policy Search

Input: Initial feasible sampler π_0, optimizer $\mathcal{A}$, inner-loop horizon T, repeat count K
Evaluate π_0 on the fixed seeds to obtain $\widehat{\mathcal{J}}_{\mathrm{val}}(\pi_0)$ and diagnostic summary d_0; set $\pi_{\mathrm{best}} \leftarrow \pi_0$
for $k = 0, 1, 2, \ldots, K_{\mathrm{outer}} - 1$ **do**
  Propose edited candidate $\tilde{\pi}_k \sim q_\psi(\cdot \mid \pi_k, d_k)$
  Validate and repair $\tilde{\pi}_k$ if needed
  Run K repeated PINN trainings and compute $\widehat{\mathcal{J}}_{\mathrm{val}}(\tilde{\pi}_k)$
  Use the critic to summarize diagnostics and produce d_{k+1}
  Continue local search from the feasible candidate: $\pi_{k+1} \leftarrow \tilde{\pi}_k$
  if $\widehat{\mathcal{J}}_{\mathrm{val}}(\tilde{\pi}_k) < \widehat{\mathcal{J}}_{\mathrm{val}}(\pi_{\mathrm{best}})$ **then**
    $\pi_{\mathrm{best}} \leftarrow \tilde{\pi}_k$
  end if
end for
Output: Best observed feasible sampler π_{best}

5 Experiments

Baselines. We compare CoDyS primarily against optimizer-centric baselines because the main question is whether learned collocation can improve the optimization regime of a strong PINN solver, not merely outperform another hand-designed sampler. Adam Kingma & Ba (2015) is the standard first-order baseline; LRA Wang et al. (2021) and NTK Wang et al. (2022b) represent loss balancing; PCGrad Yu et al. (2020), MGDA Sener & Koltun (2018), CAGrad Liu et al. (2021), and Aligned-MTL Senushkin et al. (2023) represent multi-objective gradient manipulation; MultiAdam Yao et al. (2023) and DPM Kim et al. (2021) provide adaptive and momentum-based stabilization; and DCGD Hwang & Lim (2024) is the most relevant geometry-aware optimizer for PINNs.

Table 1: Performance comparison across PDE benchmarks. Values report mean reference-based validation relative L_2 error with standard deviation (lower is better). The optimizer rows reproduce the complete baseline set and five-equation protocol of the published DCGD evaluation (Hwang & Lim, 2024). MGDA does not converge on the 5D heat equation due to severe gradient imbalance in high dimensions.

Method	Helmholtz	Burgers	Klein–Gordon	Heat (5D)	Helmholtz (3D)
Adam	0.0609 (0.0231)	0.0683 (0.0285)	0.0792 (0.0386)	0.0097 (0.0072)	0.6109 (0.2096)
LRA	0.0066 (0.0025)	0.0180 (0.0094)	0.0069 (0.0037)	0.0052 (0.0056)	0.0831 (0.0123)
NTK	0.0358 (0.0107)	0.0224 (0.0061)	0.0223 (0.0151)	0.0027 (0.0012)	0.4037 (0.2620)
PCGrad	0.0109 (0.0031)	0.0159 (0.0061)	0.0286 (0.0064)	0.0083 (0.0049)	0.2532 (0.0476)
MGDA	0.7590 (0.1180)	0.9780 (0.0462)	0.6690 (0.2790)	–	0.9883 (0.0217)
CAGrad	0.0735 (0.0390)	0.0321 (0.0063)	0.1850 (0.0301)	0.0043 (0.0016)	0.5854 (0.3032)
Aligned-MTL	0.6570 (0.0805)	0.0294 (0.0129)	0.5571 (0.1824)	0.0013 (0.0004)	0.9138 (0.0645)
MultiAdam	0.0211 (0.0032)	0.0875 (0.0303)	0.0228 (0.0038)	0.0009 (0.0007)	0.7809 (0.0031)
DCGD	0.0040 (0.0010)	0.0304 (0.0040)	0.0072 (0.0011)	0.0008 (0.0005)	0.0739 (0.0587)
CoDyS + LLaMA-3.1-8B	0.0023 (0.0005)	0.0120 (0.0057)	0.0065 (0.0020)	0.0010 (0.0011)	0.0566 (0.0042)
CoDyS + Qwen-3-Coder-30B	0.0025 (0.0006)	0.0093 (0.0049)	0.0070 (0.0018)	0.0007 (0.0004)	0.0503 (0.0067)
CoDyS + DeepSeek-Coder-V2	0.0036 (0.0013)	0.0103 (0.0060)	0.0068 (0.0016)	0.0006 (0.0005)	0.0585 (0.0085)
CoDyS + Qwen-2.5-Coder-32B	0.0028 (0.0007)	0.0097 (0.0082)	0.0068 (0.0009)	0.0005 (0.0003)	0.0589 (0.0044)
CoDyS + GPT-5 Mini	0.0027 (0.0005)	0.0095 (0.0067)	0.0066 (0.0072)	0.0005 (0.0004)	0.0583 (0.0059)
CoDyS + GPT-5 Nano	0.0034 (0.0016)	0.0099 (0.0083)	0.0068 (0.0016)	0.0006 (0.0005)	0.0526 (0.0078)

Experimental Setup. All experiments were conducted on a consumer-grade workstation with an AMD Ryzen 9 7900X CPU, 32 GB DDR5 RAM, and an NVIDIA RTX 4080 SUPER GPU (16 GB VRAM), showing that CoDyS does not require specialized infrastructure. We evaluate CoDyS with several small to medium language models, including LLaMA-3.1-8B, Qwen-3-Coder-30B, DeepSeek-Coder-V2, Qwen-2.5-Coder-32B, and GPT-5 Nano, to show that performance does not depend on a particular generator. For each PDE benchmark, candidate samplers are ranked by repeat-averaged relative L^2 error on a fixed reference-based validation grid and common training seeds. Physics losses and dual-cone statistics are auxiliary diagnostics supplied to the critic. The reported best-policy values use this same validation protocol. Additional implementation details are deferred to the Appendix.

PDEs. Table 1 combines optimizer-baseline values reported in the published DCGD evaluation (Hwang & Lim, 2024) with our CoDyS results on the same five-PDE suite. This is a protocol-aligned literature comparison, not a rerun of every optimizer in one codebase. The suite covers two- and three-dimensional Helmholtz equations, Burgers' equation, the nonlinear Klein–Gordon equation, and a five-dimensional heat equation. CoDyS matches or improves upon the reported DCGD results across the suite. A detailed Helmholtz (2D) case study is provided in Appendix A.

Discussion & Failures of Baselines. Across all benchmarks, optimizer-centric PINN baselines fail in systematic ways because their update rules act only after fixed collocation has instantiated the empirical objective. Methods such as Adam, LRA, NTK, and gradient-based multi-objective optimizers can rescale or manipulate the resulting gradients, but they cannot recover physical information that the collocation set did not represent. These failures are strongest in nonlinear, oscillatory, and higher-dimensional settings, where pseudo-random collocation yields uneven constraint enforcement, oscillatory loss trajectories, and poorly conditioned empirical gradients. CoDyS acts upstream: it searches for a PDE-specific empirical landscape whose residual and boundary signals are more useful to DCGD, reducing dual-cone violations and stabilizing the trajectory. In this sense, CoDyS *optimizes the optimizer's operating regime*: DCGD determines how to move given a pair of gradients, while CoDyS learns which physical evaluations should generate those gradients.

Direct Sampler Comparisons and Mechanistic Ablations. Table 2 compares (i) fixed, manually allocated, and residual-adaptive samplers under the same DCGD optimizer, (ii) trajectory and geometry feedback ablations, and (iii) generator–critic design ablations. All rows share the network, loss, training horizon, validation protocol, and DCGD update; full settings are reported in Appendix A.3.3. The LLM-free controls exclude agent overhead.

Table 2: LLM-free sampler baselines and mechanistic ablations under a fixed DCGD inner-loop optimizer. Values report mean reference-based validation relative L_2 error with standard deviation.

Variant	Helmholtz	Burgers	Klein–Gordon
CoDyS full: DCGD + learned generator–critic sampler	**0.0023 (0.0005)**	**0.0120 (0.0057)**	**0.0065 (0.0020)**
Sampler controls under fixed DCGD (no LLM agents)			
DCGD + default fixed sampler	0.0068 (0.0021)	0.0315 (0.0092)	0.0184 (0.0067)
DCGD + fixed interior/boundary allocation	0.0049 (0.0014)	0.0227 (0.0076)	0.0121 (0.0043)
DCGD + residual-only adaptive sampler	0.0037 (0.0010)	0.0179 (0.0061)	0.0098 (0.0031)
Feedback controls within the CoDyS outer loop (with LLM agents)			
CoDyS without geometry feedback	0.0056 (0.0019)	0.0264 (0.0085)	0.0157 (0.0052)
CoDyS without residual/boundary balance term	0.0042 (0.0013)	0.0208 (0.0070)	0.0114 (0.0038)
CoDyS with sparse trajectory feedback only	0.0039 (0.0011)	0.0186 (0.0064)	0.0102 (0.0035)
Outer-loop search controls			
Generator-only search without LLM critic	0.0045 (0.0016)	0.0239 (0.0079)	0.0136 (0.0046)
Single-LLM shared generator/critic	0.0038 (0.0012)	0.0197 (0.0068)	0.0109 (0.0036)

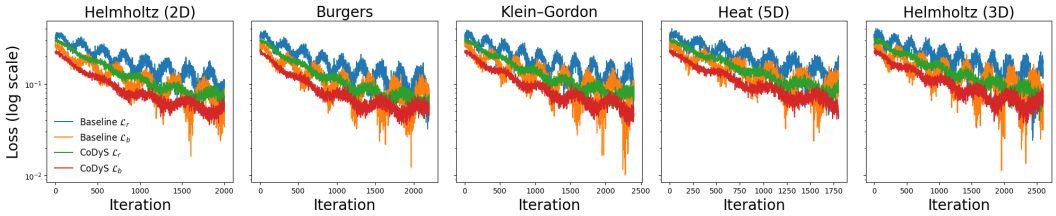

Figure 3: Mechanistic ablation under a fixed DCGD optimizer. Fixed or restricted sampling yields higher-variance, less synchronized losses, while full CoDyS produces smoother coupled dynamics.

Table 2 is the attribution counterpart to the optimizer comparison: every row uses the same DCGD update, so only the empirical landscape supplied to DCGD changes. Relative to the strongest LLM-free adaptive baseline in this study, residual-only adaptive sampling, full CoDyS reduces mean error by 37.8% on Helmholtz, 33.0% on Burgers, and 33.7% on Klein–Gordon; relative to the default fixed sampler, the reductions are 66.2%, 61.9%, and 64.7%, respectively. These results show that the gain is not DCGD alone and not adaptivity alone. It arises from jointly controlling interior and boundary sampling using trajectory-level, optimizer-aware feedback. In the second block, removing geometry feedback, residual–boundary balance terms, or dense trajectory feedback consistently degrades performance, indicating that the DCGD-conditioned diagnostics are important for shaping the sampled objective. In the third block, removing the critic or merging generator and critic into one LLM also weakens performance, showing that the generator–critic decomposition contributes beyond selecting a standard adaptive rule. Figure 3 shows the same pattern: fixed or restricted sampling yields noisier, less synchronized losses, whereas CoDyS produces smoother coupled trajectories and more stable gradient geometry.

6 Conclusions, Limitations, and Future Work

We show that learned collocation policies can control gradient geometry in PINNs and address failure modes that optimizer design alone cannot resolve. CoDyS improves stability and accuracy across nonlinear, oscillatory, and higher-dimensional PDEs by changing where constraints are enforced. Our analysis is limited to collocation-based PINNs with fixed losses and finite horizons; highly irregular engineering geometries may require more explicit geometric parameterization, and our fixed-loss setting leaves open how dynamic sampling interacts with dynamic loss-weighting methods. Extending sampler learning to initial-condition-dominated systems, operator-learning models, and long-horizon bi-level optimization remains future work.

AI USE STATEMENT

Generative AI tools were used only to improve the grammar, clarity, and readability of author-written text. They were not used to generate research ideas, hypotheses, methods, experimental designs, code, data, mathematical claims or proofs, analyses, interpretations, results, figures, or citations. The authors reviewed all AI-assisted edits and take full responsibility for the final content of this paper.

ETHICS STATEMENT

This work studies computational methods for training physics-informed neural networks on synthetic PDE benchmarks. It does not involve human participants, personal data, or the release of sensitive datasets. The experiments do not directly raise privacy, fairness, discrimination, or security concerns. The authors have reviewed and adhered to the ICLR Code of Ethics and take responsibility for the integrity and accurate reporting of this work. Applications of scientific machine learning in safety-critical or high-consequence settings should undergo appropriate domain validation and oversight before deployment.

REPRODUCIBILITY STATEMENT

The main text specifies the CoDyS search procedure, ranking objective, controlled sampler comparisons, and evaluation protocol. The appendix provides the prompts, random-seed schedule, PDE and model definitions, network architectures, optimizer and scheduler settings, collocation-point budgets, validation-grid construction, hyperparameters, convergence analysis, and runtime measurements needed to reproduce the experiments. It also documents the provenance of inherited optimizer-baseline results and distinguishes them from the newly evaluated CoDyS results. Source code is also provided as supplementary material.

REFERENCES

Dohyun Bu, Yujung Byun, and Jong-Seok Lee. Harmonized cone for feasible and non-conflict directions in training physics-informed neural networks. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=PRYl1mO1go.

Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Mitigating propagation failures in physics-informed neural networks using retain-resample-release (R3) sampling. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 7264–7302. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/daw23a.html.

Siyuan Duan, Wenyuan Wu, Peng Hu, Zhenwen Ren, Dezhong Peng, and Yuan Sun. Copinn: Cognitive physics-informed neural networks. In *International Conference on Machine Learning*, 2025.

Zhongkai Hao, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, and Jun Zhu. Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 76721–76774. Curran Associates, Inc., 2024. doi: 10.52202/079017-2442.

Youngsik Hwang and Dong-Young Lim. Dual cone gradient descent for training physics-informed neural networks. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 98563–98595. Curran Associates, Inc., 2024. doi: 10.52202/079017-3128.

Aja Khanal, Kaushik T. Ranade, Rishabh Agrawal, Kalyan S. Basu, and Apurva Narayan. Agents of diffusion: Enhancing diffusion language models with multi-agent reinforcement learning for

structured data generation. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1–20, Paphos, Cyprus, 2026. International Foundation for Autonomous Agents and Multiagent Systems. Extended Version.

Jungeun Kim, Kookjin Lee, Dongeun Lee, Sheo Yon Jhin, and Noseong Park. DPM: A novel training method for physics-informed neural networks in extrapolation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8146–8154, 2021. doi: 10.1609/aaai.v35i9.16992. URL https://doi.org/10.1609/aaai.v35i9.16992.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 26548–26560. Curran Associates, Inc., 2021.

Xiaojian Li, Yuhao Liu, and Zhengxian Liu. Physics-informed neural network based on a new adaptive gradient descent algorithm for solving partial differential equations of flow problems. *Physics of Fluids*, 35(6):063608, 06 2023. ISSN 1070-6631. doi: 10.1063/5.0151244. URL https://doi.org/10.1063/5.0151244.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 18878–18890. Curran Associates, Inc., 2021.

Qiang Liu, Mengyu Chu, and Nils Thuerey. ConFIG: Towards conflict-free training of physics-informed neural networks. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=APojAzJQiq.

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

Maricela Best McKay, Avleen Kaur, Chen Greif, and Brian Wetton. Near-optimal sketchy natural gradients for physics-informed neural networks. In *International Conference on Machine Learning*, 2025.

Johannes Muller and Marius Zeinhofer. Achieving high accuracy with PINNs via energy natural gradient descent. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 25471–25485. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/muller23b.html.

Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, 36(8):962–977, 2021. doi: 10.1111/mice.12685. URL https://doi.org/10.1111/mice.12685.

Guofei Pang and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

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

Nilo Schwencke and Cyril Furtlehner. ANaGRAM: A natural gradient relative to adapted model for efficient PINNs learning. In *International Conference on Learning Representations*, volume abs/2412.10782, 2025.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Dmitry Senushkin, Nikolay Patakin, Arseny Kuznetsov, and Anton Konushin. Independent component alignment for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20083–20093, 2023.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43 (5):A3055–A3081, 2021. doi: 10.1137/20M1318043. URL https://doi.org/10.1137/20M1318043.

Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks, 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. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2021.110768. URL https://www.sciencedirect.com/science/article/pii/S002199912100663X.

Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, 2023. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2022.115671. URL https://www.sciencedirect.com/science/article/pii/S0045782522006260.

Qingpo Wuwu, Chonghan Gao, Tianyu Chen, Yihang Huang, Yuekai Zhang, Jianing Wang, Jianxin Li, Haoyi Zhou, and Shanghang Zhang. PINNsAgent: Automated PDE surrogation with large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68143–68165. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/wuwu25a.html.

Jiachen Yao, Chang Su, Zhongkai Hao, Songming Liu, Hang Su, and Jun Zhu. MultiAdam: Parameter-wise scale-invariant optimizer for multiscale training of physics-informed neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 39702–39721. PMLR, 2023. URL https://proceedings.mlr.press/v202/yao23c.html.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5824–5836. Curran Associates, Inc., 2020.

Chenhong Zhou, Jie Chen, Zaifeng Yang, and Ching Eng Png. Dual-balancing for physics-informed neural networks. In James Kwok (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 7164–7172. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/797. URL https://doi.org/10.24963/ijcai.2025/797. Main Track.

A Appendix

A.1 Scope and Computational Budget

With DCGD used in every row of the first block, the comparison isolates the sampler contribution: the 33.0–37.8% improvement over residual-adaptive sampling cannot be attributed to changing the inner optimizer. The main optimizer table includes every optimizer baseline used by the original DCGD evaluation; methods outside that protocol, such as SOAP, would require a new optimizer-conditioned search and are not part of the present evidence. The architecture, inner-loop horizon,

batch size, and repeat protocol are matched across sampler baselines. CoDyS additionally incurs candidate rollouts and generator/critic inference, so accuracy alone is not a compute-normalized comparison. We report this cost rather than omit it: Appendix A.4.2 separates LLM generation, PINN training, and total wall-clock time for a detailed run, and Appendix A.5 reports timing across all five PDEs and local generator models. In the detailed run, generation accounts for 227.4 ± 14.6 seconds versus 3666.2 ± 68.5 seconds of PINN training per iteration, or $5.8 \pm 0.3\%$ of wall-clock time. These measurements quantify the LLM overhead, but do not establish wall-clock or API-cost superiority over simpler samplers. Separate implementations of every gradient-, causal-, and importance-based sampler under a fully matched end-to-end search budget remain outside the present comparison.

A.2 Sampling-Literature Positioning

Table 3: Positioning of CoDyS relative to representative PINN sampling approaches. "Trajectory feedback" denotes information collected across an inner training rollout rather than a pointwise acquisition score.

Family	Adapted object	Primary signal	Relation to CoDyS
Space-filling (Wu et al., 2023)	Point locations or resampling	Domain coverage	Fixed inexpensive rule
Residual/importance (Wu et al., 2023; Nabian et al., 2021)	Point density or refinement set	Current residual or loss	Prescribed acquisition statistic
R3 (Daw et al., 2023)	Retained and replaced points	Residual history	Prescribed retain–release rule
Causal training (Wang et al., 2022a)	Temporal loss emphasis	Causal progression	Specialized to temporal ordering
CoDyS	Executable joint interior–boundary sampler	Validation rollout plus trajectory diagnostics	Searches composed sampling operations

A.3 Reproducibility

This section provides all resources necessary to fully reproduce the experiments described in the **CoDyS** framework. CoDyS formulates collocation sampling for physics-informed neural networks as a bi-level learning problem, where a *sampling program generator* and a *critic* jointly optimize physics-constrained collocation strategies to stabilize gradient geometry under a fixed PINN optimizer.

Each experiment follows a structured outer-loop procedure in which natural language programs specify collocation sampling rules. These programs control how residual and boundary points are allocated across the domain. For every reported PDE benchmark, candidate samplers are ranked by repeat-averaged relative L^2 error on a fixed reference-based validation grid. Gradient alignment, balance metrics, PDE losses, and dual-cone stability indicators are recorded and fed back to the critic as diagnostic signals, but they do not enter the scalar ranking objective. The implementation and prompt templates historically call the reference grid a "test set"; operationally it is a validation set because its L^2 error is repeatedly exposed to the outer search.

This section contains:

1. The exact natural language prompts and templates used to generate collocation sampling programs.
2. A complete example of a CoDyS training run, including sampler proposals, critic feedback, optimization metrics, and resulting PINN loss trajectories for a representative PDE.

By following the provided prompts, sampling programs, and experiment logs, researchers can fully reproduce the sampling, training, and evaluation pipeline without modifying PINN architectures or optimizer implementations. This confirms that CoDyS is reproducible, model-agnostic, and deployable on consumer-grade hardware, while isolating sampler learning as the sole adaptive component of the framework.

Table 4: Summary of notation used throughout the paper.

Symbol	Description
$\Omega \subset \mathbb{R}^d$	Spatial or space–time domain of the PDE
$\partial\Omega$	Boundary of the domain Ω
d	Dimension of the domain
x	Spatial or space–time coordinate
$u(x)$	True solution of the PDE
$u(x;\theta)$	PINN approximation parameterized by θ
$\theta \in \mathbb{R}^p$	Trainable PINN parameters
$\mathcal{N}[\cdot]$	Differential operator defining the PDE
$\mathcal{B}[\cdot]$	Boundary condition operator
$f(x)$	Forcing term of the PDE
$g(x)$	Boundary condition target
$\mathcal{X}_r^t$	Residual (interior) collocation points at iteration t
$\mathcal{X}_b^t$	Boundary collocation points at iteration t
N_r, N_b	Number of residual and boundary points
π	Collocation sampling policy
π^*	Optimized sampling policy
$\mathcal{L}(\theta)$	Total PINN loss
$\mathcal{L}_r(\theta)$	PDE residual loss
$\mathcal{L}_b(\theta)$	Boundary loss
g_r^t	Residual gradient $\nabla_\theta \mathcal{L}_r(\theta_t)$
g_b^t	Boundary gradient $\nabla_\theta \mathcal{L}_b(\theta_t)$
R_t	Gradient norm ratio $\|g_r^t\|/\|g_b^t\|$
φ_t	Angle between g_r^t and g_b^t
$\mathcal{A}$	Fixed optimizer (e.g., DCGD, Adam)
η_t	Learning rate at iteration t
θ_t	PINN parameters at iteration t
T	Training horizon
$\mathcal{G}$	Set of geometrically favorable gradient configurations
$\mathcal{J}(\pi)$	Sampler objective functional
g_{dual}^t	Dual-cone–compatible update direction

A.3.1 RANDOM SEEDS AND EVALUATION PROTOCOL.

To ensure reproducibility and fair comparisons, we use a fixed, shared set of random seeds across all methods and experiments:

$$\mathcal{S} = \{111, 2222, 3333, 4444, 55555, \\ 6666, 7777, 8888, 9999, 11111\}$$

For evaluations repeated K times, the k-th repeat uses the k-th seed from this set (in the listed order). Unless otherwise noted, results use $K = 10$, so each reported mean and standard deviation is computed from K runs with distinct seeds.

A.3.2 PROMPTS

Klein-Gordon

You are an ML Python engineer. You edit ONE file: klein_gordon_sampler.py. This file is responsible ONLY for generating training points (samplers) for a PINN that solves a 1D Klein-Gordon-type PDE in (t, x).

Problem (read-only, already implemented in core code and training loop):

- Variables: (t, x) with $t \in [0, 1]$ and $x \in [0, 1]$
- Domain sampler: points inside $[0, 1] \times [0, 1]$ used for PDE residual loss
- Initial condition sampler (ICS): points on $t = 0$, $x \in [0, 1]$
- Boundary condition samplers (BCs): points on $x = 0$ and $x = 1$ (or the provided BC segments)
- The PDE residual, IC/BC losses, normalization, and test set evaluation are implemented in fixed core modules and are NOT part of the file you edit.

Your job:

- Design better sampling strategies to improve training performance, especially lowering mean L2 error on the test set.
- Create more ROBUST strategies and don't just create a naive sampler.

Hard constraints (must NOT change):

- The file must define class Klein_Gordon_sampler.
- Do not change the public API or the class name.
- Keep methods that return sampler objects with a .sample(N) method.
- The samplers must output points ONLY (shape $(N, 2)$) in coordinate order (t, x).
- Do not add new external dependencies beyond: numpy, pyDOE, scipy, torch, tqdm

Output requirements:

- Your WHOLE response is a COMPLETE sampler file as plain valid Python source code.
- Do not include any markdown fences.
- At the very top include a reasoning block as Python comments (first line: `# REASONING:`)
- Do NOT just reformat or trivially rewrite; make real, meaningful changes.
- Do NOT use slow loops in the code.

Burgers

You are an ML Python engineer. You edit ONE file: burgers_sampler_new.py. The class MUST be called Burgers_sampler_new. This file is responsible ONLY for generating training points (samplers) and the test set for a PINN that solves the 1D viscous Burgers equation. Edit the imports if needed but do NOT remove them.

Problem (read-only, already implemented in core code and training loop):

- PDE: $u_t(x, t) + u(x, t) \cdot u_x(x, t) - \nu \cdot u_{xx}(x, t) = 0$
- Domain: $(x, t) \in [-1, 1] \times [0, 1]$
- Viscosity: $\nu = 0.01/\pi$ (fixed value internally)
- Initial condition: $u(x, 0) = -\sin(\pi x)$

- Boundary conditions: $u(-1,t)=0$ and $u(1,t)=0$ for $t\in[0,1]$
- The PDE residual, initial condition loss, boundary losses, and evaluation are implemented in fixed core modules and are NOT part of the file you edit.

Training interface (fixed, must match the model):

- The training loop calls: X_ui, ui, X_ub, ub, X_f = sampler.sample(batch_size)
- X_ui is $(N_{ui},2)$ with $t=0$; ui is $(N_{ui},1)$ values $u(x,0)$
- X_ub is $(N_{ub},2)$ with $x=-1$ or $x=1$; ub is $(N_{ub},1)$ boundary values
- X_f is $(N_f,2)$ interior collocation points
- Evaluation calls: X_test, u_test = sampler.testset()

Your job:

- Design better sampling strategies to improve training performance, especially lowering mean L2 error on the test set.
- You must NOT reimplement or change the PDE, ICs, BCs, or loss; treat them as black boxes.

Hard constraints (must NOT change):

- The file must define class Burgers_sampler_new.
- Do not change the public API or the class name.
- Do not add new external dependencies beyond: numpy, pyDOE, scipy, torch, tqdm

Output requirements:

- Your WHOLE response is a COMPLETE sampler file as plain valid Python source code.
- Do not include any markdown fences.
- At the very top include a reasoning block as Python comments (first line: `# REASONING:`)
- Do NOT just reformat or trivially rewrite; make real, meaningful changes.

Helmholtz

You are an ML Python engineer. You edit ONE file: helmholtz_sampler.py. This file is responsible ONLY for generating training points (samplers) for a PINN that solves the 2D Helmholtz PDE.
Problem (read-only, already implemented in core code and training loop):

- PDE: $\Delta u(x_1,x_2)+k^2u(x_1,x_2)=0$
- Domain: $(x_1,x_2)\in[-1,1]\times[-1,1]$
- Boundary conditions: $u(x_1,x_2)=0$ on the boundary
- The PDE residual, boundary losses, and test set evaluation are implemented in fixed core modules and are NOT part of the file you edit.

Your job:

- Design better sampling strategies to improve training performance, especially lowering mean L2 error on the test set.
- You must NOT reimplement or change the PDE, BCs, or loss; treat them as black boxes.

Hard constraints (must NOT change):

- The file must define class Helmholtz_sampler.
- Do not change the public API or the class name
- Do not add new external dependencies beyond: numpy, pyDOE, scipy, torch, tqdm

Output requirements:

- Your WHOLE response is a COMPLETE sampler file as plain valid Python source code.
- Do not include any markdown fences.
- At the very top include a reasoning block as Python comments (first line: `# REASONING:`)
- Do NOT just reformat or trivially rewrite; make real, meaningful changes.
- Do NOT use slow loops in the code.

Heat 5D

You are an ML Python engineer.
You edit ONE file: heat5d_sampler.py. The class MUST be called Heat5D_sampler. This file is responsible ONLY for generating training points (samplers) and the test set for a PINN that solves the 5D Heat equation. Edit the imports if needed but do NOT remove them.
Problem (read-only, already implemented in core code and training loop):

- PDE: $u_t(x,t) - \alpha\,\Delta u(x,t) = 0$
- Spatial variables: $x = (x_1, x_2, x_3, x_4, x_5)$
- Domain: $(x,t) \in [0,1]^5 \times [0,1]$
- Diffusivity: $\alpha = 1.0$ (fixed value internally)
- Initial condition: $u(x,0) = \prod_{i=1}^{5} \sin(\pi x_i)$
- Boundary conditions: $u(x,t) = 0$ on the boundary of $[0,1]^5$ for all $t \in [0,1]$
- The PDE residual, initial condition loss, boundary losses, normalization, and evaluation are implemented in fixed core modules and are NOT part of the file you edit.

Training interface (fixed, must match the model):

- The training loop calls: X_ui, ui, X_ub, ub, X_f = sampler.sample(batch_size)
- X_ui is $(N_{ui}, 6)$ with $t = 0$; ui is $(N_{ui}, 1)$ values $u(x,0)$
- X_ub is $(N_{ub}, 6)$ points on the spatial boundary; ub is $(N_{ub}, 1)$ boundary values
- X_f is $(N_f, 6)$ interior space–time collocation points
- Evaluation calls: X_test, u_test = sampler.testset()

Your job:

- Design better sampling strategies to improve training performance in high dimension, especially lowering mean L2 error on the test set.
- Focus on robustness and sample efficiency in 5D; naive uniform sampling is unlikely to work well.
- You must NOT reimplement or change the PDE, ICs, BCs, or loss; treat them as black boxes.

Hard constraints (must NOT change):

- The file must define class Heat5D_sampler.
- Do not change the public API or the class name.
- Keep methods that return sampler objects with a `.sample(N)` method.
- The samplers must output points ONLY (shape $(N, 6)$) in coordinate order $(x_1, x_2, x_3, x_4, x_5, t)$.
- Do not add new external dependencies beyond: numpy, pyDOE, scipy, torch, tqdm

Output requirements:

- Your WHOLE response is a COMPLETE sampler file as plain valid Python source code.
- Do not include any markdown fences.
- At the very top include a reasoning block as Python comments (first line: `# REASONING:`)
- Do NOT just reformat or trivially rewrite; make real, meaningful changes.

Helmholtz 3D

You are an ML Python engineer. You edit ONE file: helmholtz_sampler.py. This file is responsible ONLY for generating training points (samplers) for a PINN that solves the 3D Helmholtz PDE.
Problem (read-only, already implemented in core code and training loop):

- PDE: $\Delta u(x_1, x_2, x_3) + k^2 u(x_1, x_2, x_3) = 0$
- Domain: $(x_1, x_2, x_3) \in [-1,1] \times [-1,1] \times [-1,1]$
- Boundary conditions: $u(x_1, x_2, x_3) = 0$ on the boundary
- The PDE residual, boundary losses, and test set evaluation are implemented in fixed core modules and are NOT part of the file you edit.

Your job:

- Design better sampling strategies to improve training performance, especially lowering mean L2 error on the test set.
- You must NOT reimplement or change the PDE, BCs, or loss; treat them as black boxes.

Hard constraints (must NOT change):

- The file must define class Helmholtz_sampler.
- Do not change the public API or the class name
- Do not add new external dependencies beyond: numpy, pyDOE, scipy, torch, tqdm

Output requirements:

- Your WHOLE response is a COMPLETE sampler file as plain valid Python source code.
- Do not include any markdown fences.
- At the very top include a reasoning block as Python comments (first line: `# REASONING:`)
- Do NOT just reformat or trivially rewrite; make real, meaningful changes.

A.3.3 Implementation Hyperparameters and Tuning Protocol

Tables 5–7 report the concrete settings exposed by the released implementation. We distinguish fixed experimental settings from quantities searched by CoDyS: the outer loop searches sampler programs, but it does not tune the PINN architecture or optimizer hyperparameters.

Outer-Loop Configuration. Table 5 summarizes the outer-loop hyperparameters. These values were selected based on preliminary experiments during method development and held fixed across all reported results.

Table 5: Outer-loop hyperparameters.

Parameter	Value	Description
`max_iters`	10	Outer-loop iterations
`max_retries`	2	Retries per iteration
`repeat`	10	Training runs per eval
`temperature`	0.2	LLM temperature
`top_p`	0.9	Nucleus sampling
`repeat_penalty`	1.1	Repetition penalty

The implementation sets the outer-loop budget to 10 proposals with at most two failed-generation retries per proposal. Local generators use temperature 0.2, top-$p = 0.9$, repetition penalty 1.1, and seed 42. Candidate evaluation uses the common seed schedule in Appendix A.1; the reported protocol uses $K = 10$ repeats. The optional convergence check uses a minimum of six observations, a five-iteration window, and relative-improvement tolerance 0.01.

Shared inner-loop configuration. The released entry point exposes one shared PINN configuration, held fixed for the CoDyS and sampling-ablation runs.

Table 6: Inner-loop (PINN) hyperparameters.

Parameter	Value	Description
Learning rate	10^{-3}	Adam base LR
Base batch size	128	Per constraint sampler unless noted below
Iterations	50,000	Training horizon
Optimizer	DCGD	Center variant
(β_1, β_2)	$(0.9, 0.999)$	Adam momentum
Adam ϵ	10^{-8}	PyTorch default
Weight decay	0	No parameter regularization
LR schedule	StepLR$(1000, 0.9)$	Applied except in the Burgers loop
Hidden width	50	Every hidden layer
Hidden depth	3	Three hidden layers
Activation	$\tanh$	All hidden layers
Initialization	Xavier normal	Zero biases
Loss weights	1	Unweighted sum of empirical MSE terms
Precision	FP32	Network and automatic differentiation

The resulting architectures are $[2, 50, 50, 50, 1]$ for Helmholtz (2D), Burgers, and Klein–Gordon; $[3, 50, 50, 50, 1]$ for Helmholtz (3D); and $[6, 50, 50, 50, 1]$ for Heat (5D). Table 7 gives the actual number of points requested from each sampler at every training iteration.

Table 7: PDE-specific batching and validation grids in the released implementation. Here $B = 128$. Counts are per training iteration; separate boundary faces each receive the stated number of points.

PDE	Residual	Initial/boundary	Validation set
Helmholtz (2D)	$10B = 1280$	$4 \times B$ boundary	100×100 grid
Burgers	$10B = 1280$	B initial, $2B$ boundary	256×101 grid
Klein–Gordon	$B = 128$	B initial, $2 \times B$ boundary	100×100 grid
Heat (5D)	$10B = 1280$	B initial, $10 \times B$ boundary	20,000 random points
Helmholtz (3D)	$10B = 1280$	$6 \times B$ boundary	50^3 grid

Hyperparameter-search disclosure. No exhaustive per-method hyperparameter sweep was run. The optimizer-baseline values in Table 1 are inherited from the published DCGD comparison and were not rerun or retuned in our codebase. Our CoDyS experiments use the architecture, learning rate, batch size, training horizon, DCGD variant, scheduler, and loss weights in Table 6 to align with that reported protocol. Table 1 should therefore be read as a protocol-aligned literature comparison, not a compute-matched reimplementation or the independently tuned optimum of every method.

A.3.4 Convergence Analysis

CoDyS optimizes sampling policies through an outer loop that proposes candidate policies and evaluates them via repeated PINN training. Since the generator model is not fine-tuned during this procedure, convergence refers to stabilization of the best-found policy under a fixed training and evaluation protocol, rather than convergence of generator parameters.

Outer-loop objective (repeat-averaged evaluation). Let π denote a sampling policy that induces collocation sets $(\mathcal{X}_r, \mathcal{X}_b)$, where $\mathcal{X}_r \subset \Omega$ contains interior points for the PDE residual and $\mathcal{X}_b \subset \partial\Omega$ contains boundary points. Let $\mathcal{A}$ denote the fixed optimizer, and let E_{val} be relative L^2 error on the fixed reference-based validation grid used by the outer loop. Each evaluation is repeated across a fixed set of random seeds $\{r_i\}_{i=1}^{K}$, where K matches the `repeat` count in the implementation. We estimate

$$\mathcal{J}_{\text{val}}(\pi) \;=\; \frac{1}{K}\sum_{i=1}^{K} E_{\text{val}}(\mathcal{A}(\pi; r_i))\,. \tag{12}$$

Lower values indicate better validation performance. The validation grid is separate from sampled training collocation batches, but the outer loop queries it repeatedly and receives the corresponding analytic or numerical reference error. Using the same seed set for all candidates implements a common-random-numbers design, reducing comparison variance. The best reported policy is $\arg\min_\pi$ over observed $\mathcal{J}_{\text{val}}(\pi)$ values.

Best-so-far tracking and reporting. Let $\tilde{\pi}_k$ denote the sampling policy proposed at outer iteration k. We track the best objective observed so far,

$$B_k \;=\; \min_{0 \le i \le k} \mathcal{J}_{\text{val}}(\tilde{\pi}_i), \tag{13}$$

which is non-increasing by construction. This yields a guarantee for the reported best policy: allocating additional outer-loop budget cannot worsen B_k, even though intermediate proposals may be worse than earlier ones. The evaluation engine reports both local comparisons (against the previous policy) and global comparisons (against the best observed policy), ensuring that improvements remain directional and cumulative over long runs.

Practical convergence and stopping rule. Beyond monotone best-so-far tracking, CoDyS uses a plateau-based stopping criterion to detect diminishing returns. Let W denote a sliding window length over outer iterations (distinct from K, the repeat count). We declare convergence when the relative improvement in best-so-far objective over the last W outer iterations falls below a tolerance ϵ:

$$\frac{B_{k-W} - B_k}{B_{k-W}} < \epsilon. \tag{14}$$

This criterion directly reflects the goal of efficient policy discovery: it triggers once additional proposals cease to yield meaningful best-so-far improvements under the controlled repeat-averaged evaluation protocol.

A.3.5 Feasibility Enforcement and Failure Recovery

Candidate policies are executed through lightweight validation wrappers that enforce interface correctness (expected shapes and types) and physical validity (support restricted to the interior domain or the appropriate boundary segments). This constrains exploration to deployable policies and prevents misleading improvements driven by invalid sampling. In addition, the engine detects and isolates failure modes (e.g., runtime errors or pathological slowdowns) and uses them as informative signals to guide subsequent proposals.

CoDyS supports long-horizon optimization by automatically archiving policy versions, logging structured outcomes, and restoring the last known working policy upon runtime errors or timeouts. This yields a self-healing search process that preserves experimental continuity and makes extended optimization runs robust to occasional invalid proposals.

A.3.6 Artifact Logging and Benchmarking

The same engine evaluates multiple generator models under identical constraints and training protocols, enabling controlled comparisons across LLM families and scales. All interactions and artifacts are recorded (policy versions, training statistics, and structured feedback), producing a provenance trail suitable for scientific auditing, ablations, and reproducibility.

A.4 Case Study: Detailed Run of Helmholtz (2D)

We provide a complete optimization trajectory from a single CoDyS run on the 2D Helmholtz benchmark using Qwen3-Coder (30B) as the generator model.

Table 8: Optimization trajectory for Helmholtz (2D) with Qwen3-Coder (30B). Each row reports relative L^2 error ℓ_2 (mean $\pm$ std over $K = 10$ repeats), final residual and boundary losses $(\mathcal{L}_r, \mathcal{L}_b)$ (means), and wall-clock training time (seconds; mean).

Iter	ℓ_2 (mean $\pm$ std)	$\mathcal{L}_r$	$\mathcal{L}_b$	Time (s)	vs. Base	Strategy
	0.00400 ± 0.00100	0.13555	1.48×10^{-4}	3156.0	–	Uniform random
1	0.00359 ± 0.00133	0.01419	2.29×10^{-5}	3545.5	-10.2%	LHS, stratified BC
2	0.00295 ± 0.00097	0.01832	1.76×10^{-5}	3572.5	-26.4%	Sobol, stratified BC
3	0.00270 ± 0.00091	0.02134	1.76×10^{-5}	3729.0	-32.5%	Halton, corner conc.
4	0.00298 ± 0.00112	0.02045	2.18×10^{-5}	3690.5	-25.6%	Sobol, corner/edge
5	0.00293 ± 0.00066	0.01924	2.81×10^{-5}	3714.0	-26.6%	Halton, determ. edges
6	0.00293 ± 0.00066	0.01924	2.81×10^{-5}	3732.5	-26.6%	(unchanged)
7	$\mathbf{0.00256 \pm 0.00056}$	**0.01115**	$\mathbf{1.46 \times 10^{-5}}$	**3626.5**	$\mathbf{-36.0\%}$	Sobol, $3\times$ corner
8	0.00276 ± 0.00072	0.01975	1.95×10^{-5}	3701.5	-30.9%	Sobol, $2\times$ corner
9	0.00274 ± 0.00098	0.01673	2.07×10^{-5}	3683.5	-31.5%	Sobol, $1\times$ corner

A.4.1 Narrative Analysis: Evolution of Sampling Strategies

This section provides a detailed account of how sampling strategies evolved over the optimization trajectory shown in Table 8, connecting observed performance changes to the gradient geometry framework developed in Section 4.

Baseline Configuration. The optimization begins with uniform random sampling for both interior and boundary collocation points. The PINN achieves a relative L^2 error of 4.00×10^{-3} (std 1.00×10^{-3}) with PDE residual loss $\mathcal{L}_r \approx 0.136$ and boundary loss $\mathcal{L}_b = 1.48 \times 10^{-4}$. The ratio $\mathcal{L}_r/\mathcal{L}_b \approx 915$ indicates severe imbalance in constraint enforcement, corresponding to a large gradient magnitude ratio $R_t = \|g_r\|/\|g_b\|$. Under such imbalance, the residual gradient dominates optimization dynamics, and the conditions required for stable dual-cone updates are unlikely to be consistently satisfied.

The standard deviation across repeated runs (1.00×10^{-3}) indicates high variability in the induced gradient configurations, as uniform random sampling produces different spatial distributions on each repeat.

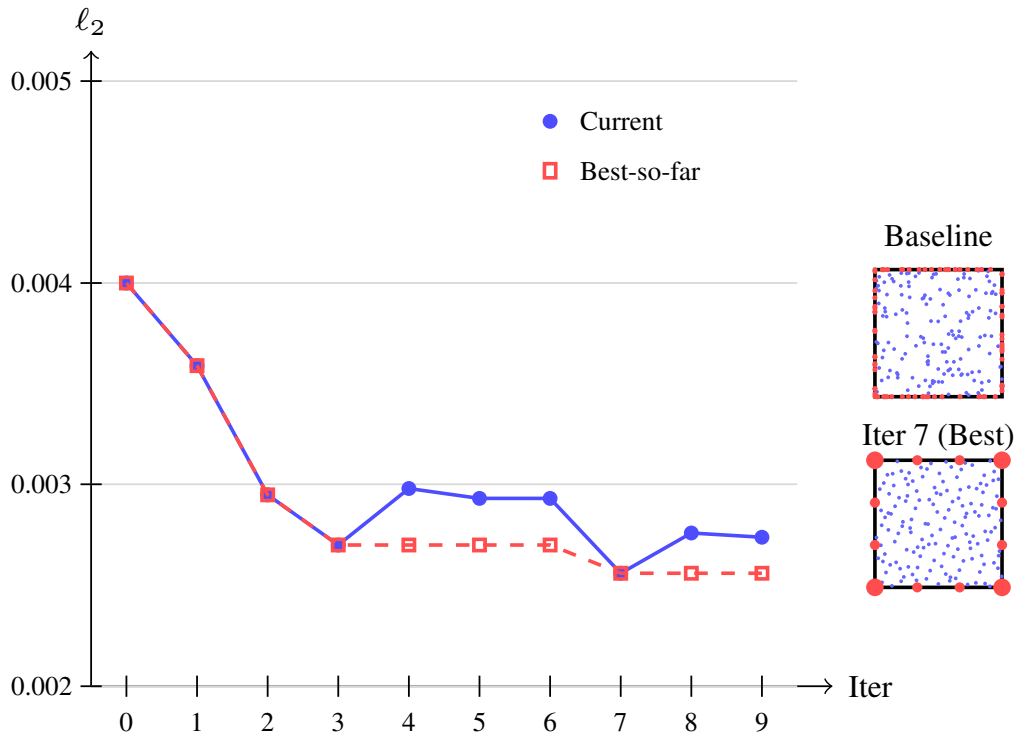

Figure 4: Optimization trajectory and collocation distributions for Helmholtz (2D). (a) Convergence trajectory. (b) Collocation distributions.

Iteration 1: Structured Sampling. Latin Hypercube Sampling (LHS) is introduced for interior points, with stratified sampling for boundary points to ensure coverage of all four edges.

The PDE residual loss decreases from ≈ 0.136 to ≈ 0.014 (about an 89.5% reduction), while the boundary loss decreases from 1.48×10^{-4} to 2.29×10^{-5}. The loss ratio becomes $\mathcal{L}_r/\mathcal{L}_b \approx 620$, indicating improved balance and a gradient configuration where R_t is closer to unity. The L^2 error improves by 10.2% to 3.59×10^{-3}.

This result indicates that the baseline's poor performance stemmed from inadequate spatial coverage rather than limitations of the PINN architecture or optimizer.

Iteration 2: Quasi-Monte Carlo Sequences. LHS is replaced with a Sobol sequence for interior collocation, providing lower-discrepancy coverage in two dimensions.

The validation L^2 error decreases by 26.4% to 2.95×10^{-3}. The PDE residual loss increases slightly (from ≈ 0.014 to ≈ 0.018) while reference-grid performance improves, suggesting that Sobol sampling better fits the validation domain.

Iteration 3: Corner Concentration. Corner concentration is introduced, with additional boundary points placed near the four corners of the domain, while the interior sampling switches from Sobol to Halton sequences.

This combination yields an L^2 error of 2.70×10^{-3}, a 32.5% improvement over baseline. The boundary loss remains low at 1.76×10^{-5}, while the loss ratio shifts to $\mathcal{L}_r/\mathcal{L}_b \approx 1210$, indicating that residual enforcement again dominates the aggregate loss despite improved test error.

Iteration 4: Corner/Edge Emphasis. Sampling is adjusted to emphasize corners and edges while reverting the interior sampling to Sobol sequences.

The L^2 error regresses from 2.70×10^{-3} to 2.98×10^{-3} (about a 10.4% increase vs. iteration 3). The boundary loss increases to 2.18×10^{-5} while the PDE residual loss remains similar ($\mathcal{L}_r \approx 0.020$).

This regression illustrates that the relationship between boundary emphasis and performance is non-monotonic: shifting boundary density and placement can move the ratio R_t away from regimes where dual-cone conditions are reliably satisfied.

Iterations 5–6: Exploration and Stagnation. Halton sequences are reintroduced with deterministic edge spacing. The L^2 error improves slightly to 2.93×10^{-3}, but remains above the best result from iteration 3. Iteration 6 produces identical statistics, indicating that the proposed modifications did not substantively change the sampling distribution.

Iteration 7: Effective Configuration. Sobol sequences are combined with moderate corner concentration (3 points per corner), drawing on elements from earlier iterations.

The L^2 error decreases to 2.56×10^{-3}, a 36.0% improvement over baseline. The PDE residual loss drops to $\mathcal{L}_r \approx 0.011$, the lowest in the trajectory, while the boundary loss decreases to 1.46×10^{-5}. The loss ratio $\mathcal{L}_r/\mathcal{L}_b \approx 760$ represents improved balance compared to baseline. The standard deviation of the L^2 error is 5.6×10^{-4}, indicating that this configuration produces more consistent outcomes across repeats than earlier strategies.

Iterations 8–9: Reduced Corner Concentration. Corner concentration is reduced to 2 points per corner (iteration 8), then 1 point per corner (iteration 9). The L^2 error increases to 2.76×10^{-3} and 2.74×10^{-3}, respectively. These results confirm that the 3-point corner concentration from iteration 7 was well-calibrated: both more aggressive boundary emphasis (iteration 4) and fewer corner points (iterations 8–9) yield worse performance.

Overall, the trajectory demonstrates a 36.0% reduction in relative L^2 error achieved purely through sampling optimization, with no modifications to the network architecture, loss formulation, or optimizer. Although the final policy can be summarized post hoc using familiar labels such as Sobol sampling and corner concentration, those labels are only an interpretable description of one learned program in a much larger policy class. The contribution of CoDyS is therefore not the rediscovery of an existing named sampler, but the outer-loop learning procedure that identifies, calibrates, and validates such coupled design choices from trajectory-level optimization feedback.

A.4.2 LLM Overhead and Wall-Clock Cost

Table 9: LLM inference overhead for the Helmholtz (2D) case study using Qwen3-Coder (30B) via Ollama. Reported times are wall-clock seconds. The LLM share is computed as $t_{\mathrm{LLM}}/(t_{\mathrm{LLM}} + t_{\mathrm{train}})$ per iteration (with $K = 10$).

Iter	t_{LLM} (s)	t_{train} (s)	t_{total} (s)	LLM share (%)
1	199.0	3545.5	3744.5	5.3
2	222.6	3572.5	3795.1	5.9
3	217.2	3729.0	3946.2	5.5
4	221.9	3690.5	3912.4	5.7
5	238.1	3714.0	3952.1	6.0
6	225.8	3732.5	3958.3	5.7
7	240.9	3626.5	3867.4	6.2
8	247.6	3701.5	3949.1	6.3
9	233.8	3683.5	3917.3	6.0
Mean ± std	227.4 ± 14.6	3666.2 ± 68.5	3893.6 ± 76.6	5.8 ± 0.3
Total (9 iters)	2046.9	32995.5	35042.4	5.8

Inference latency. Each CoDyS iteration in this case study makes a single local Ollama call to Qwen3-Coder (30B) to synthesize a new sampler. Across the nine successful iterations in Table 9, generation takes 227.4 ± 14.6 s, while the subsequent PINN training run takes 3666.2 ± 68.5 s. Thus, LLM inference contributes only $5.8 \pm 0.3\%$ of the per-iteration wall-clock time. Summed over the run, this is 34.1 min of generation overhead versus 550.0 min of training (584.0 min total).

Implications. Including these costs clarifies that CoDyS's gains come with a small, largely fixed per-iteration generation overhead. Since t_{LLM} does not scale with evaluation repeats K while train-

ing does, the LLM fraction remains modest even at higher K; at $K = 10$, the per-iteration LLM share is only $\approx 5.8\%$, making the generation cost negligible relative to the training budget.

A.5 Runtime Analysis

Table 10: Runtime breakdown for CoDyS. Inner loop reports average throughput (iter./s) and total PINN training time (seconds, averaged over 10 repeats). Outer loop reports average LLM generation time per training run (seconds) for each generator model.

	Inner Loop		Outer Loop			
PDE	**Iter./s**	**PINN (s)**	**LLaMA 3.1 8B**	**Qwen 2.5 32B**	**Qwen 3 30B**	**DeepSeek 16B**
Klein–Gordon	160.30	3433.3	133.4	350.8	179.4	48.8
Helmholtz (2D)	170.74	3491.0	105.2	315.2	194.8	42.8
Burgers	231.58	2125.9	120.5	357.2	238.4	46.4
Heat (5D)	96.73	5202.1	160.3	361.4	278.3	78.3
Helmholtz (3D)	118.07	4423.5	113.7	386.3	268.3	69.7

Table 10 reports baseline PINN training throughput (inner loop) together with the end-to-end runtime overhead introduced by sampler learning (outer loop) across PDEs and generator models. Baseline training sustains between 96 and 232 iterations per second, with throughput governed by a combination of input dimensionality, PDE operator complexity, and network width. In particular, the five-dimensional heat equation exhibits the lowest iteration rate due to its higher-dimensional input and deeper network, while Burgers' achieves the highest throughput owing to its one-dimensional spatial structure and comparatively lightweight operator.

These differences are reflected in total inner-loop runtimes, which range from approximately 2.1k seconds for Burgers' to over 5.2k seconds for the 5D heat equation. In contrast, the outer-loop costs introduced by CoDyS remain modest across all settings. Sampler evaluation and policy updates require between 40 and 390 seconds depending on the generator model, consistently accounting for a small fraction of total runtime. Even for larger generators such as Qwen 2.5 (32B) and Qwen 3 (30B), sampler learning overhead remains minor relative to PINN training, while smaller or quantized models incur negligible cost.

Importantly, CoDyS treats the PINN training loop as a black box and does not modify forward or backward passes. Consequently, its computational footprint scales independently of PDE dimensionality and network architecture, and the effective training speed remains close to baseline throughput. These results demonstrate that sampler learning based on gradient-geometry-aware feedback is computationally practical on consumer-grade hardware and that CoDyS delivers stability and accuracy gains without imposing prohibitive runtime overhead.

B Experimental Details

B.1 Models

Our model selection covers a range of parameter scales, includes both open-weight and API-accessible models, and prioritizes code-focused models given that the sampler agent must produce executable Python programs. We evaluate six models spanning 8B to 32B parameters.

Qwen3-Coder (30B). We select Qwen3-Coder as a high-capacity, code-specialized open-weight generator to evaluate CoDyS under strong program synthesis capabilities. Its training emphasis on structured code generation makes it well suited for emitting executable sampling programs with nontrivial control flow and low-discrepancy constructions. With 30B parameters, Qwen3-Coder represents the upper bound of models that can be run locally on consumer-grade GPUs with tensor parallelism or memory-efficient inference. We use greedy decoding (temperature $= 0$, top-$p = 1$) to minimize stochasticity in sampler synthesis and to ensure that observed variability arises from learning dynamics rather than generation noise. This setting provides a near-deterministic generator baseline for assessing the best-case performance of CoDyS under open-weight models.

LLaMA 3.1 Instruct (8B, FP16). To probe the lower end of the capacity spectrum, we include LLaMA 3.1 Instruct (8B) as a compact, general-purpose instruction-tuned model. This choice evaluates whether CoDyS depends on large or code-specialized generators, or whether sampler learning remains effective with limited representational capacity. The model is run in FP16 with greedy decoding to preserve numerical stability and reduce variance across runs. By holding decoding parameters fixed and minimal, this configuration isolates the effect of model capacity on the ability to propose useful sampling policies, demonstrating that CoDyS does not rely on large-scale reasoning or extensive code priors to function effectively.

DeepSeek-Coder-V2 (16B, Lite Instruct, Q4_0). We include DeepSeek-Coder-V2 under 4-bit quantization (Q4_0) to evaluate CoDyS in memory- and compute-constrained deployment scenarios. This model balances moderate parameter count with explicit code training, while quantization substantially reduces memory footprint and bandwidth requirements. We retain greedy decoding and restrict maximum generation length to prevent degradation from quantization-induced noise. This setting tests whether CoDyS remains robust when the generator operates under reduced numerical precision, reflecting practical constraints faced by researchers without access to high-memory accelerators.

Qwen2.5-Coder (32B). Qwen2.5-Coder is included to enable a controlled comparison across successive generations within the same model family. Paired with Qwen3-Coder, this isolates the impact of architectural and training improvements on sampler synthesis while holding parameter count approximately constant. Both models are run with identical decoding hyperparameters and prompt structure, allowing differences in performance to be attributed to model quality rather than inference settings. This comparison helps clarify whether CoDyS benefits primarily from scale or from improved inductive biases in newer code-focused LLMs.

GPT-5 Mini (OpenAI API). To benchmark against a high-capacity proprietary model, we include GPT-5 Mini accessed via the OpenAI API. This model provides strong reasoning and code generation capabilities without requiring local GPU resources. We use deterministic decoding (temperature $= 0$) to ensure reproducibility across API calls. Including an API-based model demonstrates that CoDyS can be deployed in environments where local compute is limited, while also providing a strong reference point for sampler quality achievable with state-of-the-art closed-weight systems.

GPT-5 Nano (OpenAI API). We pair GPT-5 Nano with GPT-5 Mini to study how sampler synthesis quality scales with model capacity under a shared inference interface. Both models use identical prompts and decoding settings, enabling a clean comparison that isolates capacity effects without confounding hardware or optimization differences. This pairing further supports the claim that CoDyS is model-agnostic: while higher-capacity models can converge more quickly or produce slightly stronger samplers, the framework remains effective even with substantially smaller generators.

B.2 Partial Differential Equations

Helmholtz Equation (2D). We consider the two-dimensional Helmholtz equation

$$\Delta u(x, y) + k^2 u(x, y) = f(x, y), \quad (x, y) \in \Omega,$$

with homogeneous Dirichlet boundary conditions $u(x, y) = 0$ on $\partial\Omega$, where $\Omega = [-1, 1]^2$. This PDE is a canonical benchmark for studying oscillatory phenomena and is widely used to stress-test PINNs due to its sensitivity to collocation placement in the presence of high-frequency spatial modes. In particular, small mismatches in interior sampling can lead to significant under-enforcement of oscillatory constraints, causing PINNs to converge to overly smooth solutions that satisfy boundary conditions but fail to resolve fine-scale structure.

We employ a manufactured solution of the form

$$u^*(x, y) = \sin(a_1 \pi x) \sin(a_2 \pi y),$$

with the corresponding forcing term

$$f(x, y) = (k^2 - a_1^2 \pi^2 - a_2^2 \pi^2) \sin(a_1 \pi x) \sin(a_2 \pi y).$$

This construction induces anisotropic oscillations across spatial directions, creating a controlled setting in which accurate PDE enforcement requires sufficiently structured interior collocation. As a result, the 2D Helmholtz equation serves as an ideal testbed for evaluating whether learned sampling policies can adapt collocation geometry to the spectral properties of the solution, rather than relying on fixed or heuristic point distributions.

Viscous Burgers' Equation. We consider the viscous Burgers' equation

$$u_t(t,x) + u(t,x)u_x(t,x) - \nu u_{xx}(t,x) = 0,$$
$$(t,x) \in [0,1] \times [-1,1]$$

with initial condition $u(0,x) = -\sin(\pi x)$ and homogeneous Dirichlet boundary conditions $u(t,-1) = u(t,1) = 0$. This equation is a standard nonlinear benchmark for PINNs, as it exhibits convection-dominated dynamics and the formation of sharp spatial gradients, even when the solution remains smooth. These features induce a highly nonuniform residual landscape in space–time, making training particularly sensitive to how collocation points are distributed.

The Burgers' equation is especially challenging for optimizer-centric methods because localized regions of steep gradients dominate the residual while large portions of the domain contribute relatively little signal. Under fixed or pseudo-random sampling, these regions are frequently underresolved, leading to imbalanced gradient contributions between the PDE residual and boundary terms and resulting in unstable or biased optimization trajectories. Consequently, this benchmark provides a stringent test of whether adaptive collocation strategies can concentrate sampling effort where nonlinear dynamics are most informative, making it well-suited for evaluating the effectiveness of CoDyS in controlling sampling-induced optimization pathologies.

Klein–Gordon Equation. We consider the nonlinear Klein–Gordon equation

$$u_{tt}(t,x) - u_{xx}(t,x) + \gamma u^k(t,x) = f(t,x),$$
$$(t,x) \in [0,1] \times [0,1]$$

with prescribed initial conditions $u(0,x) = x$, $u_t(0,x) = 0$, and Dirichlet boundary conditions on the spatial domain. This equation serves as a canonical benchmark for time-dependent nonlinear PDEs, combining second-order temporal dynamics with a nonlinear restoring force. The interaction between temporal oscillations and nonlinear coupling makes accurate enforcement of the PDE residual highly sensitive to space–time collocation placement.

From an optimization perspective, the Klein–Gordon equation is challenging because residual errors can accumulate and propagate forward in time if early-time regions are under-sampled. Fixed or pseudo-random sampling often produces irregular temporal coverage, leading to poorly conditioned gradients and unstable training dynamics, even when geometry-aware optimizers are used. As such, this benchmark is well suited for evaluating whether adaptive sampling can regularize temporal coverage, balance residual and boundary gradients, and stabilize optimization in nonlinear, second-order-in-time systems.

Heat Equation (5D). To isolate the impact of dimensionality on collocation-based PINN training, we include the five-dimensional heat equation

$$u_t(t,x) = \Delta u(t,x), \quad (t,x) \in [0,1] \times [-1,1]^5,$$

with smooth initial and boundary conditions. While the heat equation itself is linear and exhibits stable diffusive dynamics, its five-dimensional formulation poses a significant challenge for collocation methods due to measure concentration: the interior volume grows rapidly relative to the boundary, rendering uniform or pseudo-random sampling increasingly inefficient.

In this regime, residual estimates become noisy and gradient magnitudes poorly balanced, even in the absence of strong nonlinearity or oscillations. As a result, optimization instability arises primarily from sampling geometry rather than PDE stiffness. This benchmark therefore serves as a controlled test of whether adaptive collocation can mitigate the curse of dimensionality by allocating samples in a manner that preserves effective residual enforcement and stable gradient geometry in high-dimensional domains.

Helmholtz Equation (3D). We further consider the three-dimensional Helmholtz equation

$$\Delta u(x) + k^2 u(x) = f(x), \quad x \in [-1, 1]^3,$$

with homogeneous Dirichlet boundary conditions. Extending the Helmholtz equation to three spatial dimensions substantially increases the complexity of the solution manifold, as oscillatory structure must be resolved across a higher-dimensional domain.

This benchmark is particularly challenging for collocation-based PINNs because high-frequency modes occupy a small fraction of the domain volume, making them unlikely to be adequately constrained under uniform or pseudo-random sampling. As a result, residual gradients are often dominated by low-frequency components, leading to inaccurate solutions despite stable optimization. The three-dimensional Helmholtz equation therefore provides a stringent test of whether learned sampling policies can jointly address oscillatory physics and increased dimensionality by improving interior coverage and enforcing residual constraints where spectral resolution is most critical.

B.3 PINN Architectures and Training Configuration

All experiments use fully connected feed-forward neural networks (MLPs) with elementwise $\tanh$ activations. For each PDE, the PINN represents a scalar field $u(\cdot;\theta)$ that maps the coordinate input to the predicted solution value, with inputs given by the problem coordinates (e.g., (x, y) for 2D Helmholtz and (t, x) for 1D-in-space time-dependent PDEs). For time-dependent PDEs, time is treated as an additional input coordinate and automatic differentiation is used to obtain the required temporal and spatial derivatives. We use a single network per PDE with no domain decomposition or separate subnetworks. In the fixed-DCGD sampler comparison, collocation sampling is the only changed algorithmic component.

Architectures by dimensionality. All PDEs use three hidden layers of width 50. The input dimension changes with the PDE coordinates, giving $[2, 50, 50, 50, 1]$ for Helmholtz (2D), Burgers, and Klein–Gordon; $[3, 50, 50, 50, 1]$ for Helmholtz (3D); and $[6, 50, 50, 50, 1]$ for Heat (5D), where the latter contains five spatial coordinates and time. Hidden layers use $\tanh$ activations; weights use Xavier-normal initialization and biases are initialized to zero. Helmholtz, Klein–Gordon, and Heat coordinates are standardized using the mean and standard deviation of a uniform distribution over their domains; Burgers uses the physical coordinates directly. The complete training and batching settings are reported in Appendix A.3.3.

Loss construction. Each PINN is trained by minimizing the standard composite objective

$$\mathcal{L}(\theta) = \lambda_r \mathcal{L}_r(\theta) + \lambda_b \mathcal{L}_b(\theta),$$

where $\mathcal{L}_r$ enforces the PDE residual at interior collocation points and $\mathcal{L}_b$ enforces boundary (and when applicable, initial) constraints at boundary collocation points. We set $\lambda_r = \lambda_b = 1$ throughout to isolate the effect of sampling from loss reweighting. Residual and boundary terms are computed as empirical averages over the collocation sets sampled at each iteration; all derivatives in $\mathcal{L}_r$ and $\mathcal{L}_b$ are obtained via automatic differentiation.

Initialization and numerical setup. In our CoDyS experiments, networks use Glorot normal initialization and full-precision training, with derivatives computed by automatic differentiation. Architectures, losses, and training horizons follow the corresponding settings reported in the DCGD benchmark.

Optimizers, schedules, and training protocol. The optimizer-baseline values in Table 1 are taken from the published DCGD study rather than produced by a unified reimplementation. We evaluate CoDyS on the same benchmark suite with DCGD as the fixed inner-loop optimizer. Its update at iteration t is

$$\theta_{t+1} = \mathcal{A}\big(\theta_t; \mathcal{X}_r^t, \mathcal{X}_b^t\big), \qquad (\mathcal{X}_r^t, \mathcal{X}_b^t) \sim \pi,$$

where $\mathcal{A} = \mathrm{DCGD}$ is fixed and π denotes the collocation sampling policy. Across CoDyS runs, the architecture, objective, optimizer, training horizon, and evaluation protocol are fixed, making the learned sampler the only adaptive component.

B.4 Error Metric

We evaluate solution accuracy using the relative L^2 error defined as

$$\text{Relative } L^2 \text{ error} = \frac{\left(\sum_{i=1}^{N} |u(x_i;\theta) - u^*(x_i)|^2\right)^{1/2}}{\left(\sum_{i=1}^{N} |u^*(x_i)|^2\right)^{1/2}},$$

where $\{x_i\}_{i=1}^{N}$ denotes the fixed reference-based validation grid, u^* is the analytic solution when available, and $u(\cdot;\theta)$ is the PINN prediction. For PDEs without closed-form solutions, we use high-resolution numerical solutions as references. This relative L^2 error is repeatedly used to rank sampler programs during outer-loop search and is also the metric reported for the selected policy. The reference values are not injected as supervised pointwise targets into the PINN training loss or sampler code, but their use for selection means the reported grid is ultimately a validation set.

This metric directly reflects the global approximation quality of the learned solution and is standard in the PINN literature.

B.5 Baselines

Adam. Adam Kingma & Ba (2015) is included as the default optimizer used in most PINN implementations. Its adaptive moment estimation improves robustness to noisy and heterogeneous gradients, particularly in early training. However, Adam treats the composite PINN loss as a single objective and does not account for conflicts between PDE residual and boundary gradients. As such, it serves as a strong but non-geometry-aware baseline that exposes failure modes arising purely from fixed collocation and unbalanced gradient contributions.

Learning Rate Annealing (LRA). Learning rate annealing Wang et al. (2021) represents a class of loss-balancing methods that rescale update magnitudes based on gradient statistics. LRA mitigates dominance of a single loss term by adaptively adjusting step sizes, but it does not modify gradient directions or address misalignment between objectives. Including LRA allows us to test whether stabilizing training via magnitude control alone is sufficient under fixed collocation sampling.

Neural Tangent Kernel (NTK) Balancing. NTK-based balancing Wang et al. (2022b) leverages spectral properties of the neural tangent kernel to equalize convergence rates across loss terms in function space. This provides a theoretically grounded approach to loss reweighting that operates independently of parameter-space gradient geometry. NTK balancing is included to assess whether function-space convergence alignment compensates for sampling-induced gradient instability.

PCGrad. PCGrad Yu et al. (2020) explicitly projects gradients to remove conflicting components when objectives disagree. Originally developed for multi-task learning, PCGrad is a natural baseline for PINNs, where residual and boundary losses frequently produce opposing gradients. Its inclusion evaluates whether resolving conflicts at the gradient level can stabilize training when collocation sampling remains fixed.

MGDA. MGDA Sener & Koltun (2018) formulates training as a true multi-objective optimization problem and computes Pareto-stationary descent directions via constrained optimization. This method provides a principled mechanism for jointly optimizing residual and boundary losses without collapsing them into a scalar objective. MGDA is included to test whether Pareto-optimal descent alone can overcome sampling-induced imbalance.

CAGrad. CAGrad Liu et al. (2021) extends gradient projection approaches by explicitly trading off conflict avoidance and descent magnitude. Unlike PCGrad, which removes conflicting components, CAGrad seeks directions that minimize conflict while preserving progress. This baseline probes whether soft conflict-averse geometry is sufficient to stabilize PINNs under heterogeneous gradients.

Aligned-MTL. Aligned-MTL Senushkin et al. (2023) reduces multi-task optimization instability by aligning the principal components of the task-gradient matrix and improving its condition number. This method is particularly relevant for PINNs, where effective training requires residual and boundary gradients to cooperate. Including Aligned-MTL allows us to evaluate whether enforcing alignment at the update level compensates for poor collocation-induced gradient geometry.

MultiAdam. MultiAdam Yao et al. (2023) generalizes Adam to multi-objective settings by maintaining separate adaptive moments for each loss term. This improves robustness to scale disparities between residual and boundary gradients while preserving computational efficiency. MultiAdam tests whether per-objective adaptivity alone stabilizes training without explicit geometric reasoning.

Dynamic Pulling Method (DPM). DPM Kim et al. (2021) dynamically manipulates the update when the PDE residual exceeds a prescribed threshold and its gradient conflicts with the data-loss gradient. The resulting correction prioritizes reduction of the physics residual while minimally perturbing the original update. This baseline represents residual-prioritizing gradient manipulation for PINN training.

Dual Cone Gradient Descent (DCGD). DCGD Hwang & Lim (2024) is a geometry-aware optimizer specifically designed for PINNs. By restricting updates to a dual-cone region where both residual and boundary losses decrease simultaneously, DCGD provides theoretical guarantees against conflicting descent directions. It constitutes the strongest optimizer-centric baseline and serves as a critical reference point for CoDyS, which instead targets the source of gradient geometry by learning collocation distributions that induce favorable gradient configurations. In the main experiments, DCGD is also the fixed inner-loop optimizer used by CoDyS. Accordingly, CoDyS should be interpreted as augmenting DCGD with a learned sampling policy, not as replacing the optimizer.

C Supplementary Derivation

C.1 Sampling Dependence of Gradient Geometry

Proof. Fix an iterate θ_t and let $(\mathcal{X}_r, \mathcal{X}_b) \sim \pi$ denote the collocation sets drawn at iteration t. Define the (empirical) residual and boundary gradients as measurable maps of the sampled sets,

$$g_r \;:=\; \nabla_\theta \mathcal{L}_r(\theta_t; \mathcal{X}_r), \qquad g_b \;:=\; \nabla_\theta \mathcal{L}_b(\theta_t; \mathcal{X}_b).$$

Since $(\mathcal{X}_r, \mathcal{X}_b)$ is random under π, the pair (g_r, g_b) is a random element in $\mathbb{R}^p \times \mathbb{R}^p$.

Now define the geometry statistics (as functions of (g_r, g_b)):

$$R_t \;:=\; \frac{\|g_r\|}{\|g_b\|}, \qquad C_t \;:=\; \frac{\langle g_r, g_b\rangle}{\|g_r\|\|g_b\|} \;=\; \cos\varphi_t,$$

on the event $\{\|g_r\| > 0, \|g_b\| > 0\}$ (which holds almost surely in practice; otherwise set R_t, C_t by convention). As (R_t, C_t) is the composition of measurable functions with the random pair (g_r, g_b), it is a random vector whose law is induced by π.

Let $\mathcal{G} \subset \mathbb{R}_+ \times [-1, 1]$ denote the set of geometry configurations that satisfy the dual-cone conditions (e.g., $C_t \geq 0$ or $C_t < 0$ with the corresponding admissible range for R_t as in Hwang & Lim (2024)). Define the indicator of a dual-cone-compatible gradient configuration

$$\mathbf{1}_{\mathcal{G}}(R_t, C_t) \;:=\; \begin{cases} 1, & (R_t, C_t) \in \mathcal{G}, \\ 0, & (R_t, C_t) \notin \mathcal{G}. \end{cases}$$

Then the probability of satisfying the dual-cone conditions under policy π is

$$\mathbb{P}_\pi[(R_t, C_t) \in \mathcal{G}] \;=\; \mathbb{E}_{(\mathcal{X}_r, \mathcal{X}_b)\sim\pi}[\mathbf{1}_{\mathcal{G}}(R_t, C_t)]\,.$$

Since (R_t, C_t) depends on $(\mathcal{X}_r, \mathcal{X}_b)$ only through the induced gradients (g_r, g_b), the right-hand side is fully determined by the collocation distribution π. Therefore, both the distributions of R_t and $\cos\varphi_t$ and the probability that the dual-cone conditions hold depend on π. This derivation supports the motivating observation in the main text; it is not claimed as an independent theoretical contribution. □