REINFORCEMENT LEARNING AGENT FOR PINN OPTIMIZER CHAINS

Rustam Gabdrakhmanov, Dmitry Gusarov, Daniel Ezhov, Alexander Hvatov
AI Institute
ITMO University
Saint-Petersburg, 197101, Russia
{rrgabdrakhmanov,gusdmitr,doezhov,alex_hvatov}@itmo.ru

ABSTRACT

Physics-Informed Neural Networks (PINNs) face fundamental optimization challenges where optimizer choice dramatically outweighs architecture selection in importance. Although recent optimization chaining strategies achieve transformative performance gains, they require extensive manual tuning and domain expertise. We introduce a reinforcement-learning framework for automatic optimizer selection in PINNs, learning dynamic switching policies from training dynamics. Our approach represents optimization states through autoencoder-compressed loss landscapes, enabling the RL agent to discover connections between landscape geometry and optimal optimizer choice. This framework-agnostic method automates the construction of high-performance optimizer chains without manual intervention. Our proposed agent is evaluated on the PINNacle benchmark, where it demonstrates consistently strong results across PDE problems. We also show that the optimizer sequences automatically constructed by the agent achieve performance comparable to manually designed optimizer chains, eliminating problem-specific tuning. By transforming PINN optimization from an expert-driven process to an automated one, this work addresses a critical barrier to broader adoption of physics-informed machine learning.

1 INTRODUCTION

Physics-informed neural networks are already a popular approach for real-world problems. However, the setup and tuning burden remains high for practitioners who are not deeply familiar with mathematical modeling and machine learning. In this setting, one must tune not only the numerical aspects of the PDE solution but also the machine-learning training procedure Wang et al. (2023). Several tools are intended to simplify PINN training, such as DeepXDE Lu et al. (2021), JINNs Gangloff & Jouvin (2024), and JAXPI Wang et al. (2025). Although this is a good start, these tools do not yet provide a fully out-of-the-box solution to challenging real-world modeling problems.

Recent work shows that architecture is not an important choice for PINNs once the minimal (i.e., rather low) number of neurons is reached Urbán et al. (2025). The major game changers are the point sampling strategy and optimizer choice for most architectures Kiyani et al. (2025).

Point sampling has reached sophisticated manual and automated strategies. The main reason to adapt sampling is that PINNs are not easily approximated on a uniform grid—there is a tight connection between Korobov spaces, nonuniform Fourier approximation, and PINNs Matsubara & Yaguchi (2025). However, this introduces a contradiction: we cannot conduct new theoretical research on sampling every time a new PINN use case arises. Therefore, practical approximation strategies have emerged Lau et al. (2024), including adaptive strategies based on reinforcement learning Song (2025).

In contrast to point sampling, the choice of optimizer is unavoidable in machine learning. A key motivation for changing optimizers during training is the ill-posedness of the PINN optimization problem Rathore et al. (2024), where the condition number κ may be on the order of 10^6 to 10^8, making naive optimization ineffective.

The ill-posedness arises for several reasons. (I) It is hard to determine the proper loss ratio between the PDE operator residual, boundary conditions, and initial conditions Wang et al. (2024). This means that at a given step, it is unclear whether it is better to focus on the operator residual over the domain, the constraints, or both. (II) The relationship between the training loss and the solution error with respect to the ground truth can be highly nonlinear Zeinhofer et al. (2024). Moreover, in some cases, there is no reliable ground truth because numerical solvers can also be inaccurate. (III) There is no good "initial guess." Obtaining one often requires a classical numerical solver, which can reduce the practical advantage of PINNs. Finally, transferring learned architectures across problems remains challenging Mustajab et al. (2024).

Hand-designed optimizer chains can substantially improve PINN training quality Taylor et al. (2022). When combined with more sophisticated adaptive optimizers, the gains can be even larger Rathore et al. (2024). For example, Adam+L-BFGS can reduce errors by $14\times$, while Adam+L-BFGS+SSBFGS can yield up to $1000\times$ improvements (as reported for the loss value). Since the loss is not necessarily well aligned with the L_2 solution error, these magnitudes should be interpreted with caution.

Optimal switching strategies remain problem-specific and manually tuned. This is not unique to PINNs; for example, SWATS Keskar & Socher (2017) switches from Adam to SGD during CNN training, and multi-stage optimization has been studied more broadly for deep networks Tahmassebi et al. (2018). However, these approaches still require substantial pre-tuning and expert judgment. Although recent work demonstrates dramatic improvements from optimizer sequences, selecting the switching points and the sequence itself is typically a manual, problem-specific process. Reinforcement learning offers a principled approach to learn these decisions directly from training dynamics.

In this paper, we propose an RL-based optimizer-chain constructor we refer to as RL-PINN-OC. Our main **goal** is to demonstrate empirically that optimizer choice can matter more than architecture choice in PINNs. More specifically, we investigate whether the loss landscape predicts the best optimizer to use. We represent the loss landscape using autoencoders Elhamod & Karpatne (2023). In principle, the method applies to any PINN framework, since it relies only on the weight-update history to derive a compact loss-surface representation. We evaluate the approach in our own implementation (to avoid dependence on a single framework) and show that reinforcement learning can improve performance on the PINNacle benchmark Lau et al. (2024).

Contribution: - We show that reinforcement learning can discover connections between the loss landscape and optimizer choice.
- We propose a framework-agnostic RL agent to build optimizer chains for PINNs.
- We evaluate the method on the PINNacle benchmark using both loss reduction and l_2-error metrics.

Limitations: - While we provide a proof of concept, generalization remains an open question (e.g., how often the agent must be retrained and how well it transfers across tasks).
- Generic autoencoder representations may miss physics-specific structure; nevertheless, they provide a framework-agnostic starting point for studying optimizer–landscape interactions.
- The RL agent introduces an additional training procedure on top of standard PINN training and can be time-consuming; transfer learning may mitigate this.

Data and code are available via the anonymized repository `https://anonymous.4open.science/r/ICLR_pinnacle_exps_private-25B9/`

2 PINN Problem statement

We consider the general setting in which the physical system of interest is described by a partial differential equation posed on a spatial domain $\Omega \subset \mathbb{R}^d$ over a time horizon $[0, T]$. The strong form of the problem reads

$$\mathcal{N}[\, u(\boldsymbol{x}, t)\,] = 0, \qquad \boldsymbol{x} \in \Omega, \quad t \in [0, T], \tag{1}$$

where $\mathcal{N}[\cdot]$ is a, possibly nonlinear, differential operator that encodes the physics of the problem. The equation is supplemented by boundary conditions

$$\mathcal{B}[\,u(\boldsymbol{x},t)\,] = g(\boldsymbol{x},t), \qquad \boldsymbol{x} \in \partial\Omega, \quad t \in [0,T], \tag{2}$$

with $\mathcal{B}[\cdot]$ denoting the boundary operator and g the prescribed boundary data, and by an initial condition

$$u(\boldsymbol{x},0) = u_0(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega. \tag{3}$$

Together, equation 1–equation 3 define a well-posed initial-boundary value problem whose exact solution $u(\boldsymbol{x},t)$ we seek to approximate.

Neural Network Approximation In the Physics-Informed Neural Network (PINN) framework, the unknown solution u is replaced by a deep neural network $\hat{u}(\boldsymbol{x},t;\,\boldsymbol{\theta})$ whose trainable parameters — weights and biases collected in the vector $\boldsymbol{\theta} \in \mathbb{R}^p$ — are determined so that the network satisfies the governing equations in the mean-squared-error sense. A distinguishing feature of the formulation adopted here is that no observational or experimental data are used; the network is trained exclusively from the physics encoded in the differential operator, the boundary conditions, and the initial condition.

Loss Function To measure how well a given parameter vector $\boldsymbol{\theta}$ satisfies physics, we introduce three residual-based loss terms. The first penalizes the PDE residual in a set of N_r collocation points $\{\boldsymbol{x}_r^i, t_r^i\}_{i=1}^{N_r}$ sampled within the domain:

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_r}\sum_{i=1}^{N_r}\big|\,\mathcal{N}[\,\hat{u}(\boldsymbol{x}_r^i, t_r^i;\,\boldsymbol{\theta})\,]\big|^2. \tag{4}$$

The second enforces the boundary conditions at N_b points $\{\boldsymbol{x}_b^i, t_b^i\}_{i=1}^{N_b}$ located on $\partial\Omega$:

$$\mathcal{L}_b(\boldsymbol{\theta}) = \frac{1}{N_b}\sum_{i=1}^{N_b}\big|\,\mathcal{B}[\,\hat{u}(\boldsymbol{x}_b^i, t_b^i;\,\boldsymbol{\theta})\,] - g(\boldsymbol{x}_b^i, t_b^i)\big|^2. \tag{5}$$

The third measures the discrepancy between the network output and the prescribed initial state at N_0 points $\{\boldsymbol{x}_0^i\}_{i=1}^{N_0}$:

$$\mathcal{L}_0(\boldsymbol{\theta}) = \frac{1}{N_0}\sum_{i=1}^{N_0}\big|\,\hat{u}(\boldsymbol{x}_0^i, 0;\,\boldsymbol{\theta}) - u_0(\boldsymbol{x}_0^i)\big|^2. \tag{6}$$

The composite loss function is then defined as the weighted sum

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_r\,\mathcal{L}_r(\boldsymbol{\theta}) \;+\; \lambda_b\,\mathcal{L}_b(\boldsymbol{\theta}) \;+\; \lambda_0\,\mathcal{L}_0(\boldsymbol{\theta}), \tag{7}$$

where $\lambda_r, \lambda_b, \lambda_0 > 0$ are user-specified coefficients that control the relative importance of each contribution. The training task thus reduces to finding parameters $\boldsymbol{\theta}$ that minimize $\mathcal{L}(\boldsymbol{\theta})$.

Sequential Optimizer Chaining It is well known that the loss landscape of PINNs is highly non-convex and that no single optimization algorithm is universally superior across all stages of training. First-order stochastic methods, such as Adam, are effective at rapidly traversing large, flat regions of the loss surface during early training. However, they often stagnate near local minima or saddle points. Quasi-Newton methods, such as L-BFGS, exploit local curvature information and can achieve much tighter convergence, but they are sensitive to the quality of the initial guess. They may fail if started far from a reasonable basin of attraction.

To combine the complementary strengths of different optimizers, we propose a sequential chaining strategy. The core idea is simple: rather than committing to a single optimizer for the entire training process, we define an ordered chain of K optimizers $\{\mathcal{O}_k\}_{k=1}^{K}$ and execute them one after another. Crucially, the terminal parameters produced by each optimizer are saved and passed directly as the

initial guess to the next optimizer in the chain. In this way, each successive stage inherits and refines the solution found by its predecessor.

Formally, the optimization problem is stated as follows. Given an initial parameter vector $\boldsymbol{\theta}^{(0)}$ (obtained, for instance, via Xavier or Glorot initialization), find the optimal parameters

$$\boldsymbol{\theta}^\star = \big(\mathcal{O}_K \circ \mathcal{O}_{K-1} \circ \cdots \circ \mathcal{O}_1\big)\big(\boldsymbol{\theta}^{(0)}\big), \tag{8}$$

where the composition symbol denotes sequential application: the output of $\mathcal{O}_k$ serves as the input of $\mathcal{O}_{k+1}$.

The chaining continues stage by stage. At the beginning of stage k, the parameters are initialized from the result of the preceding stage:

$$\boldsymbol{\theta}_k^{(0)} = \begin{cases} \boldsymbol{\theta}^{(0)}, & k = 1, \\ \boldsymbol{\theta}_{k-1}^\star, & k \geq 2. \end{cases} \tag{9}$$

Here $\boldsymbol{\theta}_{k-1}^\star$ denotes the converged, or budget-exhausted, parameters from stage $k-1$. The k-th optimizer then minimizes the same composite loss $\mathcal{L}$ starting from $\boldsymbol{\theta}_k^{(0)}$, using its own set of hyperparameters (learning rate, batch size, momentum, and so on):

$$\boldsymbol{\theta}_k^\star = \mathcal{O}_k\big(\boldsymbol{\theta}_k^{(0)};\ \mathcal{L}\big). \tag{10}$$

After all K stages have been completed, the final optimized parameters are given by $\boldsymbol{\theta}^\star = \boldsymbol{\theta}_K^\star$.

Each stage k is executed subject to a set of termination conditions. In our setup, a stage (and, if applicable, the whole episode) is terminated when any of the following criteria is met: (i) the error metric reaches a prescribed tolerance, (ii) the maximum chain length is reached, (iii) the loss exceeds a predefined upper bound (used as a divergence safeguard), or (iv) an early-stopping rule indicates no further progress. Formally, we stop at stage k when

$$e(\boldsymbol{\theta}_k) \leq \varepsilon \ \text{ or } \ k \geq K_{\max} \ \text{ or } \ \mathcal{L}(\boldsymbol{\theta}_k) > \mathcal{L}_{\max} \ \text{ or } \ \mathrm{EarlyStop}\big(\{\mathcal{L}\}_{\mathrm{recent}}\big) = \mathrm{true}. \tag{11}$$

Here $e(\cdot)$ denotes the evaluation error (e.g., MSE against the reference solution), ε is the target tolerance, $K_{\max} = 10$ is the maximum allowed chain length, and $\mathcal{L}_{\max}$ is a safety threshold used to detect divergence. $\mathrm{EarlyStop}(\cdot)$ denotes an early-stopping criterion computed from the recent loss history (e.g., no improvement over a fixed patience window).

This two-stage transition can be depicted schematically as

$$\underbrace{\boldsymbol{\theta}_1^\star = \mathrm{Adam}\big(\boldsymbol{\theta}^{(0)};\ \mathcal{L}\big)}_{\text{Stage 1: first-order, stochastic}} \ \longrightarrow \ \underbrace{\boldsymbol{\theta}^\star = \text{L-BFGS}\big(\boldsymbol{\theta}_1^\star;\ \mathcal{L}\big)}_{\text{Stage 2: quasi-Newton refinement}}. \tag{12}$$

The framework is not restricted to two stages. More generally, the chain may incorporate an arbitrary sequence of first-order methods (e.g., SGD, Adam, AdamW), second-order or quasi-Newton methods (e.g., L-BFGS, Newton-CG), and learning-rate scheduling or warm restart strategies. The modular structure of the formulation makes it easy to insert additional stages, adjust stopping criteria per stage, or replace individual optimizers without modifying the rest of the pipeline. The key invariant maintained throughout is the weight-transfer mechanism expressed in equation equation 9: the knowledge accumulated by each optimizer, encoded in the network weights, is never discarded but instead passed forward to the next stage as a warm start.

3 RL Problem statement

We formulate the construction of optimizer chains for PINN training as a Markov decision process (MDP) and solve it using reinforcement learning. Instead of committing to a single optimizer

throughout training, an agent makes a sequence of decisions, selecting the next optimization stage (optimizer type and its hyperparameters) based on the current optimization landscape. One RL step corresponds to running the chosen optimizer for a fixed budget (e.g., a number of iterations or epochs), after which the environment returns an updated state and a reward. This stage-wise view is a natural fit for optimizer chaining: the agent's actions directly determine the training trajectory and therefore shape the subsequent loss landscape.

State (Landscape visualization) Scheme 1 illustrates the methodology for constructing a neural network loss landscape by mapping the weight trajectory of a trained model to a two-dimensional latent space via an autoencoder. A uniform grid is generated within this latent representation, and each grid point is inversely mapped back to the parameter space to reconstruct the corresponding weight configuration. The loss function is then evaluated for these reconstructed parameters, producing a dense sampling of the surrounding optimization space. This approach, as described in Elhamod & Karpatne (2023), enables visualization of the topology of the loss landscape in the vicinity of the actual training trajectory, facilitating identification of local minima, saddle points, and plateau regions, and thus providing deeper insights into the complexity and structure of the optimization problem.

It should be emphasized that the dimensionality of the latent space is not restricted to two. A higher-dimensional latent space can capture richer structure in the optimization dynamics and provide the agent with a more faithful representation of the underlying loss geometry. We adopt a two-dimensional latent representation in this work. This choice significantly reduces the computational cost of constructing the latent loss landscape and enables fast updates during training. In addition, the 2D case remains easy to interpret and visualize, thereby simplifying the qualitative analysis of the learned behavior and the evolving optimization dynamics.

Action (Optimizer). There are many optimization algorithms, such as LBFGS, PSO, and Adam. In addition, each method has input parameters, such as the number of epochs, learning rate, etc. The selected optimizer determines the time, benchmarks the results, and, in certain situations, whether a solution is found. Furthermore, to obtain better results, it is useful to use not just one algorithm, but a chain of optimizers Taylor et al. (2022).

Environment The environment is the PINN training procedure itself. At each step, it (i) builds the state from the current parameters, (ii) applies the optimizer stage selected by the agent, (iii) updates the parameters, and (iv) reconstructs the next state. Because the state depends on the evolving training trajectory, the agent's decisions influence both the immediate optimization outcome and the geometry observed in subsequent steps.

The interaction process is organized as follows. Based on the current PINN parameters, an environment state is formed, characterized by a loss landscape. Next, the agent selects an action, specifically the optimizer configuration. After this, the environment applies the selected optimization method to the network parameters, updates the model, changes the loss landscape, and generates a new environment state. Upon completion of this stage, the environment calculates the cumulative error of the approximate solution, including the residual of the differential operator and the errors of the boundary and initial conditions.

Thus, the environment is not a single model-optimization step but a dynamic PINN training process, in which the choice of optimizer affects the subsequent loss landscape and control strategy.

Reward The reward is calculated based on the change in the cumulative RMSE of the PINN solution from one optimization step to the next. In our approach, the RMSE is computed between the PINN predictions and the exact solution of the corresponding PDE, providing a physically interpretable measure of approximation accuracy.

The total error is defined as the weighted sum of the corresponding components:

$$E_t = \lambda_{\text{op}} \, \text{RMSE}_{\text{op}}^{(t)} + \lambda_{\text{bc}} \, \text{RMSE}_{\text{bc}}^{(t)}, \tag{13}$$

where λ_{op} and λ_{bc} are the balancing coefficients.

The reward is defined in a different form, namely, as a reduction in the cumulative error relative to its value at the previous step. In other words, the agent receives a positive reward if the selected

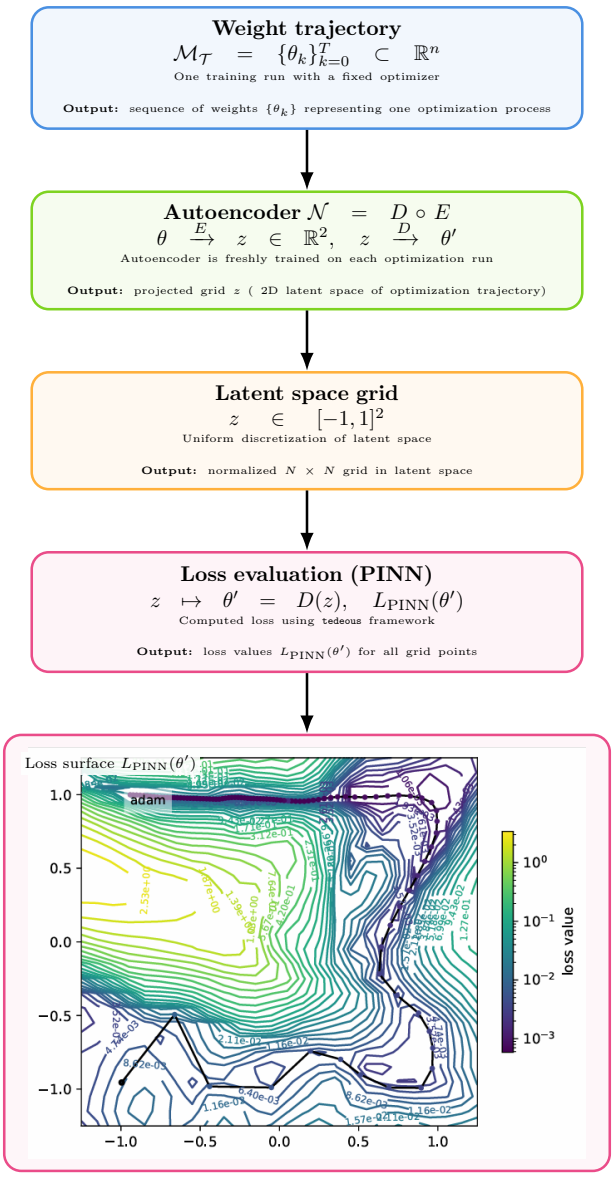

Figure 1: Scheme of loss landscape visualization pipeline.

optimizer leads to a decrease in error, and a negative reward if the error increases

$$r_t = E_{t-1} - E_t. \tag{14}$$

Using a difference form allows the agent to consider not only the absolute value of the error but also the dynamics of the optimization process.

Agent The RL agent selects the next optimization stage based on the current state. Because each environment step is computationally expensive (it runs an entire optimizer stage), we restrict the action space by considering a limited set of optimizers and discretizing continuous hyperparameters into finite choices (see App. 1). We use a DQN-based agent with an ϵ-greedy exploration strategy. Concretely, our implementation follows a Dueling Double-DQN (Wang et al. (2016), van Hasselt et al. (2015)) setup trained with a prioritized replay buffer Schaul et al. (2016), and employs a multi-head value function: one head selects the optimizer class, while additional head(s) select optimizer-specific hyperparameter choices conditioned on the selected optimizer. We also use a soft Watkins-style

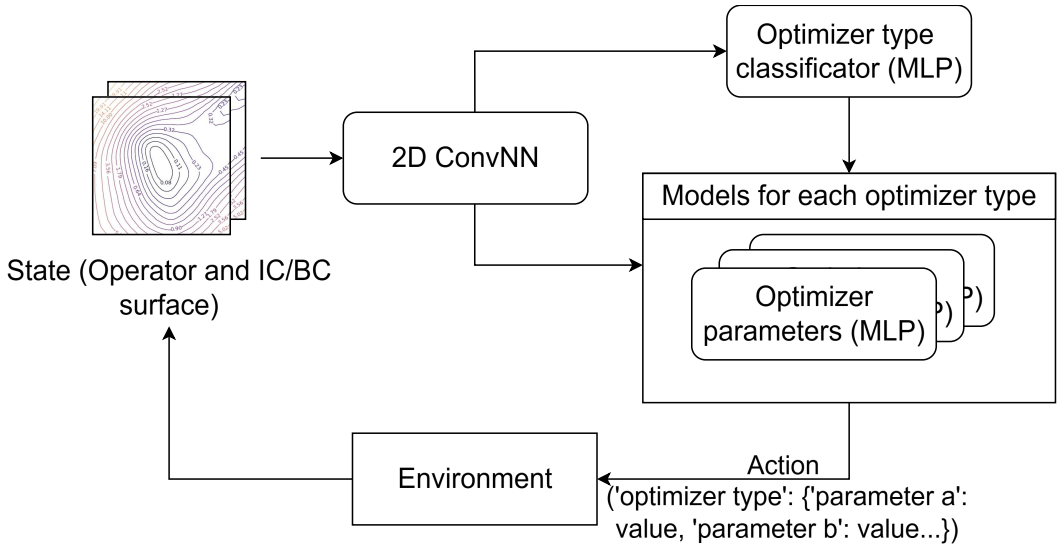

Figure 2: Scheme of RL agent cycle.

Kapturowski et al. (2022) target construction: multi-step returns are aggregated with λ-weights, and when intermediate actions deviate from the greedy policy, their contribution is softly downweighted instead of being cut off. This yields more stable targets under ϵ-greedy exploration. Since the state is represented as a structured 2D tensor (loss-landscape map with correlated spatial structure), we use a convolutional encoder (Figure 2).

Replay buffer warm-up Before training the agent, we populate the replay buffer with randomly generated transitions. Specifically, we perform approximately 10000 random interactions with the environment, where actions are sampled uniformly from the discrete action space. This pre-filling strategy addresses two key issues: (i) it prevents the agent from being trained on an empty or highly correlated buffer, and (ii) it ensures that the initial optimization process explores a sufficiently wide range of possible actions. After the buffer is initialized, we proceed with standard off-policy DQN updates while continuing to collect new transitions online.

Once the buffer is initialized, the training proceeds with the standard DQN optimization loop. To balance exploration and exploitation, we employ an ϵ-greedy policy with an exponentially decaying schedule, using the hyperparameters in Appendix A.

Stopping criterion The learning episode ends once a predefined stopping criterion is met. Specifically, we monitor the mean squared error (MSE) between the predicted PINN solution and the reference (ground-truth) solution of the differential equation. When this error falls below a given tolerance threshold, the optimization process is considered successful, and the chain construction halts. If the criterion is not met within the maximum allowed number of steps, the episode is marked as unsuccessful.

4 Experiments

The main goal of our experiments is twofold. First, we aim to empirically support the observation that, for PINNs, the choice of optimization strategy has a stronger impact on solution quality than architectural modifications or alternative loss-function variants. Second, we evaluate whether optimizer chains discovered by an RL policy can outperform single-optimizer training under the same computational budget.

Benchmark and baselines. We evaluate on the PINNacle benchmark and report our results along with the reference PINNacle baselines. For comparison, we use vanilla PINN training, reweighting/sampling strategies, optimizer variants, and alternative loss functions, as reported in the PINNacle

benchmark results. In contrast, the RL-PINN-OC column reports our method under the same problem settings. This enables a direct comparison to established PINNacle baselines across PDE types.

Action space. Throughout both agent training and evaluation, the agent operates in the same discrete action space. Concretely, it selects among Adam, L-BFGS, and PSO, where continuous hyperparameters are discretized into predefined sets:

- **Adam:** learning rate in $\{10^{-2}, 10^{-3}, 10^{-4}\}$ and stage length in $\{100, 1000, 2500\}$ epochs.
- **L-BFGS:** learning rate in $\{1, 5\times10^{-1}, 10^{-1}\}$ and stage length in $\{100, 500, 1500\}$ epochs.
- **PSO:** learning rate in $\{0, 10^{-3}, 10^{-4}\}$ and stage length in $\{100, 200, 300\}$ epochs.

Protocol. For each PDE problem, we first train a DQN-based RL agent using a warm-start phase to initialize the replay buffer with random transitions. We then continue training the agent while collecting additional experience online. After training, we freeze the trained agent and evaluate it by running a final PINN training episode under a fixed total budget of 7000 epochs. During this episode, the agent sequentially selects optimizer stages; once the cumulative number of epochs reaches the budget, the resulting optimizer chain is fully determined, and we record the final PINN metrics obtained at the end of executing this chain. To estimate variability, the final agent-driven PINN training is repeated across 10 random seeds, and we report mean and standard deviation in Tables 2–3 (MSE and relative error, respectively).

Evaluation metrics. We report mean squared error (MSE) and relative error (L2RE) as complementary measures of solution quality. MSE reflects pointwise discrepancy relative to the reference solution, while L2RE summarizes normalized error, which is comparable across problems of different scales.

Results overview. Tables 1 and 2 summarize the main results and place our RL-constructed chains (RL-PINN-OC) in the context of PINNacle baseline families. Overall, RL-PINN-OC achieves competitive performance across several PDE settings and, in particular, demonstrates substantial gains on problems where switching optimizers is beneficial, supporting the hypothesis that optimizer-level decisions dominate many other design choices in PINN training.

Table 1: MSE (mean (std)) for main experiments.

		Vanilla Loss		Reweighting/Sampling			Optimizer
Problem	Setting	PINN	PINN-w	LRA	NTK	RAR	MultiAdam
Burgers ($\cdot10^{-4}$)	1d-C	**0.79(0.18)**	2.64(0.87)	3.03(2.62)	1.30(0.52)	5.78(6.31)	9.68(5.51)
Poisson ($\cdot10^{-2}$)	2d-C	11.7(0.30)	0.03(0.01)	0.72(1.00)	**0.005(0.005)**	11.9(0.26)	0.02(0.009)
Poisson ($\cdot10^{-2}$)	2d-CG	12.8(0.10)	0.12(0.02)	0.06(0.02)	**0.007(0.003)**	13.2(0.32)	2.73(1.92)
Wave ($\cdot10^{-2}$)	1d-C	11.1(3.66)	2.54(0.16)	4.08(0.43)	0.30(0.05)	9.07(0.60)	0.47(0.13)
Chaotic	KS	1.16(0.003)	1.11(0.05)	1.04(0.006)	1.06(0.01)	1.16(0.002)	1.05(0.01)

		Loss Functions				Architecture	Meta Algorithm
Problem	Setting	gPINN	vPINN	LAAF	GAAF	FBPINN	RL-PINN-OC
Burgers ($\cdot10^{-4}$)	1d-C	177(55.8)	51.3(19.0)	1.80(1.35)	3.00(1.56)	153(103)	8.80(1.66)
Poisson ($\cdot10^{-2}$)	2d-C	11.5(0.62)	4.86(0.44)	13.9(0.57)	9.38(1.91)	0.08(0.02)	2.60(0.37)
Poisson ($\cdot10^{-2}$)	2d-CG	19.8(0.23)	2.50(0.04)	7.67(0.27)	17.7(8.70)	0.05(0.010)	7.88(1.34)
Wave ($\cdot10^{-2}$)	1d-C	9.66(0.58)	61.7(11.9)	6.03(0.29)	14.8(4.44)	13.9(1.97)	**0.03(0.06)**
Chaotic	KS	1.12(0.009)	1.05(0.003)	1.16(0.004)	1.14(0.02)	1.16(0.05)	**0.76(0.03)**

5 DISCUSSION

Our experiments support the view that optimizer chains have a pronounced effect on PINN training quality. In several problems, the RL-constructed chain reduces error by more than an order of magnitude relative to Adam alone, with the improvement achieved solely by changing the optimization strategy while keeping the architecture, loss function, and collocation scheme fixed. This observation is consistent with recent findings by Kiyani et al. (2025) and Urbán et al. (2025): once a minimum network capacity is reached, the optimizer becomes the dominant hyperparameter, outweighing architecture selection by a wide margin. Indeed, the PINNacle benchmark shows that methods

Table 2: L2RE (mean (std)) for main experiments.

		Vanilla Loss		Reweighting/Sampling			Optimizer
Problem	Setting	PINN	PINN-w	LRA	NTK	RAR	MultiAdam
Burgers ($\cdot 10^{-2}$)	1d-C	**0.95(0.064)**	1.88(0.40)	1.35(0.26)	1.30(0.17)	1.35(0.47)	2.64(0.57)
Poisson ($\cdot 10^{-1}$)	2d-C	7.40(0.055)	0.31(0.051)	0.78(0.75)	**0.13(0.082)**	7.48(0.10)	0.25(0.064)
Poisson ($\cdot 10^{-1}$)	2d-CG	5.45(0.047)	0.45(0.064)	0.26(0.055)	**0.13(0.050)**	5.60(0.082)	2.46(1.07)
Wave ($\cdot 10^{-1}$)	1d-C	5.87(0.92)	2.78(0.089)	3.49(0.20)	0.94(0.091)	5.40(0.17)	1.15(0.19)
Chaotic	KS	0.94(0.0009)	0.90(0.030)	**0.86(0.0035)**	0.86(0.0033)	0.94(0.0009)	0.87(0.0084)

		Loss Functions				Architecture	Meta Algorithm
Problem	Setting	gPINN	vPINN	LAAF	GAAF	FBPINN	RL-PINN-OC
Burgers ($\cdot 10^{-2}$)	1d-C	14.2(1.98)	4.02(0.64)	1.40(0.37)	1.95(0.83)	3.75(0.97)	4.83(0.41)
Poisson ($\cdot 10^{-1}$)	2d-C	7.35(0.21)	4.60(0.14)	7.67(0.14)	6.57(0.40)	0.50(0.047)	3.14(0.24)
Poisson ($\cdot 10^{-1}$)	2d-CG	7.31(0.028)	2.45(0.051)	4.04(0.10)	7.09(2.12)	0.32(0.062)	4.18(0.35)
Wave ($\cdot 10^{-1}$)	1d-C	5.60(0.17)	14.1(1.30)	4.38(0.14)	6.82(1.08)	6.55(0.49)	**0.20(0.19)**
Chaotic	KS	0.94(0.0061)	0.89(0.0099)	0.94(0.0032)	0.94(0.0099)	0.98(0.034)	1.02(0.019)

that modify only the loss function or the architecture (gPINN, vPINN, FBPINN) can even degrade performance on certain problems. In contrast, optimizer-level interventions such as MultiAdam or our RL agent yield more consistent gains across equation types.

A noteworthy aspect of the learned chains is that the agent discovers optimizer transitions that resemble, but do not exactly replicate, hand-designed sequences. The classical Adam→L-BFGS pipeline is a strong baseline. However, the agent occasionally inserts additional first-order stages or adjusts the switching point in ways that a human expert would be unlikely to try without exhaustive search. This suggests that the autoencoder-compressed loss landscape carries sufficient geometric information for the agent to identify when the current optimizer has exhausted its usefulness. For example, when curvature changes indicate a transition from a flat plateau to a narrow valley. At the same time, the agent does not achieve the best result on every problem. This is expected, since the agent was not specifically tuned for each equation, underscoring the need for a systematic generalization study.

The most important open question is the transferability between equations. In the current setup, the agent is trained and evaluated in the same problem class. A practical deployment would require training the agent on a diverse set of equations and testing it on unseen PDEs, following a train/test split over problem types rather than random seeds. Preliminary observations suggest that the landscape features the agent relies on, i.e., loss plateau length, gradient-norm decay rate, curvature anisotropy, are not equation-specific, which gives reason to expect positive transfer. Verifying this hypothesis rigorously, including measuring how often the agent must be retrained when the PDE family changes, is the natural next step.

6 Conclusion

We have presented the first reinforcement-learning framework for automatically constructing an optimizer chain in Physics-Informed Neural Networks. By representing optimization states via autoencoder-compressed loss landscapes and training a DQN-based agent to select optimizers dynamically, we eliminate the need for manual, problem-specific tuning of optimizer sequences. Experiments on the PINNacle benchmark demonstrate that the automatically constructed chains achieve results comparable to, and in several cases surpassing, hand-designed baselines, with particularly great improvements on hyperbolic problems. These findings reinforce the growing evidence that, in PINN training, the optimization strategy has a greater impact on design choices than on network architecture, and that automating these choices is both feasible and beneficial.

Future work will focus on training the agent across multiple PDE families simultaneously and evaluating zero-shot transfer to unseen equations, with the long-term goal of providing a fully automated, equation-agnostic optimization backend for physics-informed machine learning.

ACKNOWLEDGMENTS

This work supported by the Ministry of Economic Development of the Russian Federation (IGK 000000C313925P4C0002), agreement No139-15-2025-010

REFERENCES

Mohannad Elhamod and Anuj Karpatne. Neuro-visualizer: An auto-encoder-based loss landscape visualization method. *arXiv preprint arXiv:2309.14601*, 2023.

Hugo Gangloff and Nicolas Jouvin. jinns: a jax library for physics-informed neural networks. *arXiv preprint arXiv:2412.14132*, 2024.

Steven Kapturowski, Víctor Campos, Ray Jiang, Nemanja Rakićević, Hado van Hasselt, Charles Blundell, and Adrià Puigdomènech Badia. Human-level atari 200x faster, 2022. URL https://arxiv.org/abs/2209.07550.

Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

Elham Kiyani, Khemraj Shukla, Jorge F Urbán, Jérôme Darbon, and George Em Karniadakis. Which optimizer works best for physics-informed neural networks and kolmogorov-arnold networks? *arXiv preprint arXiv:2501.16371*, 2025.

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

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. doi: 10.1137/19M1274067.

Takashi Matsubara and Takaharu Yaguchi. Number theoretic accelerated learning of physics-informed neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 595–603, 2025.

Abdul Hannan Mustajab, Hao Lyu, Zarghaam Rizvi, and Frank Wuttke. Physics-informed neural networks for high-frequency and multi-scale problems using transfer learning. *Applied Sciences*, 14(8):3204, 2024.

Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in training pinns: A loss landscape perspective. *arXiv preprint arXiv:2402.01868*, 2024.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016. URL https://arxiv.org/abs/1511.05952.

Zhenao Song. Rl-pinns: Reinforcement learning-driven adaptive sampling for efficient training of pinns. *arXiv preprint arXiv:2504.12949*, 2025.

Amirhessam Tahmassebi, Amir H Gandomi, Simon Fong, Anke Meyer-Baese, and Simon Y Foo. Multi-stage optimization of a deep model: A case study on ground motion modeling. *PloS one*, 13 (9):e0203829, 2018.

John Taylor, Wenyi Wang, Biswajit Bala, and Tomasz Bednarz. Optimizing the optimizer for data driven deep neural networks and physics informed neural networks. *arXiv preprint arXiv:2205.07430*, 2022.

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

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. URL https://arxiv.org/abs/1509.06461.

Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An expert's guide to training physics-informed neural networks. *arXiv preprint arXiv:2308.08468*, 2023.

Sifan Wang, Ananyae Kumar Bhartari, Bowen Li, and Paris Perdikaris. Gradient alignment in physics-informed neural networks: A second-order optimization perspective. *arXiv preprint arXiv:2502.00604*, 2025.

Yong Wang, Yanzhong Yao, Jiawei Guo, and Zhiming Gao. A practical pinn framework for multi-scale problems with multi-magnitude loss terms. *Journal of Computational Physics*, 510:113112, 2024.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2016. URL `https://arxiv.org/abs/1511.06581`.

Marius Zeinhofer, Rami Masri, and Kent-André Mardal. A unified framework for the error analysis of physics-informed neural networks. *IMA Journal of Numerical Analysis*, pp. drae081, 2024.

A WARM-START HYPERPARAMETERS FOR DQN TRAINING

Table 3: Warm-start hyperparameters for DQN training.

Parameter	**Value**
Number of random transitions (buffer pre-fill)	10000
ϵ_{start}	0.5
ϵ_{end}	0.05
ϵ_{decay}	100
Target network transitions reinit	2000