HEAP: Hyper Extended A-PDHG Operator for Constrained High-dim PDEs

Mingquan Feng [*1] **Weixin Liao** [*1] **Yixin Huang** [1] **Yifan Fu** [1] **Qifu Zheng** [1] **Junchi Yan** [1]

Abstract

Neural operators have emerged as a promising approach for high-dimensional partial differential equations (PDEs). However, existing neural operators often have difficulty in dealing with constrained PDEs, which is a practical setting where the solution must satisfy additional equality or inequality constraints beyond the governing equations. To close this gap, we propose a novel neural operator, Hyper Extended Adaptive PDHG (HEAP) for constrained high-dim PDEs, where the learned operator evolves in the parameter space of PDEs. We first show that the evolution operator learning can be formulated as a quadratic programming (QP) problem, then unroll the adaptive primal-dual hybrid gradient (A-PDHG) algorithm as a QP-solver into the neural operator architecture. It allows to improve efficiency while retaining theoretical guarantees of the constrained optimization. Empirical results on a variety of high-dim PDEs show that HEAP outperforms the state-of-the-art neural operator model.

1. Introduction

Partial differential equations (PDEs) are widely used across various disciplines in science, engineering, economics, and finance (Strauss, 2007; Folland, 2020). Since many PDEs lack analytical solutions, numerical techniques like finite element methods (Huebner et al., 2001) and recently neural network-based solvers (Blechschmidt & Ernst, 2021) have been developed. Although these approaches have witnessed significant progress, they face the curse of dimensionality, where the number of required data points grows exponentially as the dimension d of the independent variables increases (Bellman, 1966).

To tackle the dimensionality challenge, recent advances in learning-based solvers often try to leverage the specific structure of PDE (Han et al., 2017; Wang et al., 2022; Yu et al., 2018) and develop memory-efficient training techniques (Hu et al., 2024; Shang et al., 2023). These methods have shown empirical success on a variety of PDEs. However, they are designed to handle specific instances, meaning that they require expensive retraining from scratch whenever there are changes in the initial or boundary conditions. Neural operators (Gaby et al., 2024; Gaby & Ye, 2024) overcome these issues by learning the evolution operator of the solution function. The training is solely based on the PDE residual loss, no needing spatial discretization or solving the PDE directly. Once trained, the model (ideally) generalizes to new initial conditions with a single inference pass.

Despite such progress, a notable bottleneck (we also note the early efforts date back to (Lagaris et al., 1998)) to neural PDE solvers is the (in)equality constraints, which are common in both real-world scenarios and numerical modeling, e.g. heat diffusion with temperature bounds and the Euler equation with entropy conditions. Simple methods incorporating these constraints can result in unstable training, slow convergence, and violation of the constraints (Wang et al., 2021). While more advanced techniques have been proposed to handle constraints in either soft penalty (Wang et al., 2024) or hard encoding (Lu et al., 2021b), these methods come with either significant computational costs or network architecture restrictions and are limited to low-dimensional PDEs as discussed in (Feng et al., 2025).

In this work, we propose a novel quadratic programming-based neural operator, HEAP, for constrained high-dimensional PDEs, which can efficiently handle constraints in high-dimensional PDEs beyond existing SOTA neural operators. We first show that the evolution operator in the parameter space of constrained PDEs can be formulated as a QP problem, where the QP constraints correspond to the quadratic linear constraints in the original space. We then unroll the adaptive primal-dual hybrid gradient (A-PDHG) algorithm into the neural operator architecture to improve computational efficiency. Moreover, we theoretically prove the approximation capability of the proposed method and provide empirical results on a variety of high-dim PDEs to demonstrate that HEAP outperforms existing SOTA neural operator model. **The contributions are as follows:**

[*]Equal contribution [1]Sch. of Computer Science & Sch. of Artificial Intelligence, Shanghai Jiao Tong University, China. Correspondence to: Junchi Yan <yanjunchi@sjtu.edu.cn>. This work was partly supported by NSFC (62222607, 623B1009). Code available at GitHub.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

- First time to identify and formulate the evolution operator in the parameter space of constrained high-dimensional PDEs as a QP optimization problem.
- Propose a hyper-extended A-PDHG operator architecture for QP problem end-to-end learning and solving, with theoretical justification.
- Solve various types of PDEs on 5-20 dimensions with various constraints accurately and stably, generalizing across initial conditions.

2. Related Work and Preliminaries

Recent neural PDE solvers have shown the potential to overcome the limitations of traditional numerical solvers, especially for high-dim and constrained problems.

2.1. From Low-dim to High-dim Neural PDEs Solvers

The neural-network-based approaches for low dimensional PDEs can be mainly divided into two categories: physics-informed network and data-driven operator(Karniadakis et al., 2021). Specifically, Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019) leverage neural networks to approximate the solution of a PDE and enforce boundary loss and PDE residual loss at a few selected points. This approach has various extensions such as weak form (De Ryck et al., 2024), adaptive sample weight (McClenny & Braga-Neto, 2020), and integral loss (Saleh et al., 2023). These methods may suffer from slow convergence and re-training requirements for new initial conditions. Alternatively, DeepONet (Chen & Chen, 1995; Lu et al., 2021a) leverages the universal approximation theorem of infinite-dimensional operators and directly fits the PDE solution operators in a data-driven manner, solving a family of PDEs with a single model. However, this approach requires a large amount of data for training and may not generalize well to new initial conditions. Physics-informed neural operators (PINO) (Li et al., 2024) is a hybrid approach incorporating data and PDE residuals in the loss function. There exist many neural operators with other architectures, such as Fourier transform layer (Li et al., 2020), and graph neural networks (GNNs)(Alet et al., 2019). The existing GNN-based models describe irregular spacial grids and their relative position as graph (Lötzsch et al., 2022; Horie & Mitsume, 2022; Bryutkin et al., 2024).

For high-dimensional PDEs, the curse of dimensionality poses a challenge, and specialized methods are designed which mainly fall in the physics-informed category. (Han et al., 2018; 2017) proposed the DeepBSDE, reforming a special class of hyperbolic PDEs as backward stochastic differential equations (BSDEs) by the Feynman-Kac formula. The deep Ritz method (Yu et al., 2018) solves high-dim variation PDEs by minimizing the energy functional of the PDE. For general form PDEs, Zang et al. (2020) proposed an adversarial network that solves PDEs. Hu et al. (2024) designed stochastic dimension gradient descent to reduce memory. Shi et al. designs an efficient amortized algorithm for differential operations in solving high-dim PDEs.

Another approach is to learn the solution operator, e.g. the mapping from initial conditions to the solution functions. The pioneering work is (Gaby et al., 2024), which learns the high-dimensional PDE solution operator by reducing the solution operator to the evolution of the parameters of a reduced-order model (a neural network) and introduces another network to learn the evolution in discrete time steps by minimizing PDE residual. The second work (Gaby & Ye, 2024) extends the first work to continuous evolution by adapting a Neural ODE (NODE) (Chen et al., 2018) framework, improving the sample efficiency. We follow this line of work and propose a novel neural operator that can handle constraints in high-dimensional PDEs.

2.2. Neural PDE Solvers with Constraints

When a PDE admits multiple weak or classical solutions, additional constraints—either equalities or inequalities—act as *regularisers* that single out the physically (or financially) meaningful branch of the solution manifold.

Initial and boundary conditions are the most common *equality* constraints, yet they may be partially unknown or insufficient to guarantee uniqueness. In such cases *inequality* constraints can still be derived from first principles. A celebrated example is the three–dimensional Navier–Stokes equation, whose uniqueness remains a Millennium Prize Problem; imposing an upper bound on the total kinetic energy is one standard way to discard non-physical blow-up solutions (Fefferman, 2006). Similarly, the multi–asset Black–Scholes equation lacks a unique solution in practice; no-arbitrage bounds on the option's *theta* restore well-posedness and are routinely used in quantitative finance (Hull & Basu, 2016). We visualise this effect in Fig. 3, where distinct constraint sets lead to markedly different Black–Scholes solution surfaces.

A few recent AI4PDE studies have begun to encode inequality information, e.g. derivative bounds or convex constraints, into physics–informed losses (Hoshisashi et al., 2024; Moro & Chamon, 2025). However, these methods rely on dense spatial discretisation and have only been demonstrated on low–dimensional ($d \leq 3$) problems. Scalability to the high–dim regime ($d \geq 10$) targeted in this work remains open; HEAP addresses this gap by reformulating the constraint enforcement as a differentiable QP layer and unrolling an adaptive PDHG solver inside the neural operator.

Previous works incorporate constraints into neural solvers in two ways: soft and hard. Soft methods introduce penalty

terms into the objective. For instance, physics-informed neural networks (PINNs) (Raissi et al., 2019) apply initial and boundary conditions as penalty terms. Other examples include enforcing turbulence models (List et al., 2022), complex geometries (Wang et al., 2024), and spectral representation (Du et al., 2023). However, in many real-world cases, the constraints need to be strictly enforced which is also the focus of this paper. Hard constraints can be implemented in various ways, e.g. encoding structure of constraints into network architecture (Lu et al., 2021b; Richter-Powell et al., 2022), using an implicit constraint layer (Négiar et al., 2022), incorporating numerical solvers into the network (Saad et al., 2022; Chalapathi et al., 2024). However, these methods are limited to low-dim PDEs and do not scale well to high-dim problems. In our work, we propose the first soft-constrained end-to-end neural solver for high-dimensional PDEs.

2.3. Genreal Formulation of Constrained PDEs

We consider a nonlinear PDE initial value problem with linear constraints in its general form:

$$\begin{cases} \partial_t u(\mathbf{x},t) = F[u](\mathbf{x},t), & \mathbf{x}\in\Omega,\ t\in(0,T], \\ u(\mathbf{x},0) = g(\mathbf{x}), & \mathbf{x}\in\Omega, \\ H[u](\mathbf{x},t) \geq 0 & \mathbf{x}\in\Omega_H,\ t\in(0,T] \end{cases} \tag{1}$$

where $u(\mathbf{x},t)$ is the solution function, F is a possible nonlinear operator, $g(\mathbf{x})$ is the initial condition, $\Omega\subset\mathbb{R}^d$ is the spatial domain, and T is the time horizon. In addition to the PDE, we have inequality constraints $H[u]\geq b$, where H is a linear operator. The constraints are defined on the subdomain Ω_H. The constraints might be rooted in the multi-solution nature of some PDEs, associated with physical laws, or numerical stability requirements.

3. Methods

3.1. Problem Formulation

Evolution Operator in the Parameter Space. Under the high-dimensional settings, it is usually infeasible to numerically calculate the PDE solution u directly. In this paper, we approximately surrogate the solution function $u(\cdot,t)$ with a neural network ansatz $u_\theta(\cdot)$, and learn the evolution of the parameters θ starting from an initial parameter θ_0. The evolution is governed by a hyper-operator V_w, which takes the current parameter θ_t and outputs its temporal derivative, i.e. $\dot\theta_t = V_w(\theta_t)$. In the definition of the constraint PDEs, we use the formulation in Equation. 1.

The overall target is to fit the hyper-operator V_w minimizing the expected PDE residual over a distribution $\mathcal{D}$ of θ_0 and under the constraints. Formally, we drive the following models for solving:

$$\begin{aligned} \min_{w} \quad & \mathop{\mathbb{E}}_{\theta_0\sim\mathcal{D}} \int_0^T \left\| \nabla_\theta u_{\theta_t}\cdot\dot\theta_t - F[u_{\theta_t}] \right\|^2_{L^2(\Omega)} \mathrm{d}t \\ \text{s.t.} \quad & H[u_{\theta_t}](\mathbf{x},t)\geq \mathbf{0}, \quad \mathbf{x}\in\Omega_H,\ t\in(0,T], \\ & \dot\theta_t = V_w(\theta_t), \quad t\in(0,T]. \end{aligned} \tag{2}$$

In this formulation, the PDE operator learning is converted to the parameter evolution operator learning, which is a linearly constrained nonlinear optimization problem.

Approximated QP Operator. To design the architecture of V_w in a principled way, we exploit the quadratic structure of Eq. 2 and approximately reformulate V_w as a QP problem w.r.t. $\dot\theta_t$. A general QP problem can be formulated as:

$$\min_{\mathbf{x}\in[\mathbf{l},\mathbf{u}]} \quad \frac{1}{2}\mathbf{x}^\top\mathbf{Q}\mathbf{x}+\mathbf{c}^\top\mathbf{x}, \quad \text{s.t.} \quad \mathbf{A}\mathbf{x}\geq\mathbf{b}, \tag{3}$$

where $\mathbf{Q}$ is a symmetric, positive semi-definite matrix. V_w can be implemented as a QP solver, after three approximation steps. Firstly, the $L^2(\Omega)$ norm in the objective function and Ω_A in the constraint are approximated by a Monte Carlo sampling, i.e. evaluation over sets of randomly sampled points $\mathbf{X},\mathbf{X}_H$ from Ω,Ω_H respectively. Secondly, u_{θ_t} in constraint A is replaced by its first-order Taylor expansion with small time step Δt, i.e. $u_{\theta_t}+\nabla_\theta u_{\theta_t}\cdot\dot\theta_t\Delta t$. Finally, we add lower and upper bounds $\mathbf{l},\mathbf{u}$ of $\dot\theta_t$ to the constraints to enhance the numerical stability. The resulting operator, termed as V_w^{QP}, leads to a new optimization problem:

$$\begin{aligned} V_w^{QP}(\theta_t) = \mathop{\mathrm{argmin}}_{\dot\theta_t\in[\mathbf{l},\mathbf{u}]} \quad & \left\| \nabla_\theta u_{\theta_t}(\mathbf{X})\dot\theta_t - F[u_{\theta_t}](\mathbf{X}) \right\|^2 \\ \text{s.t.} \quad & H[u_{\theta_t}+\nabla_\theta u_{\theta_t}\cdot\dot\theta_t\Delta t](\mathbf{X}_H)\geq 0. \end{aligned} \tag{4}$$

The problem above is a QP instance, by keeping the bound $(\mathbf{l},\mathbf{u})$ and assembling the components $(\mathbf{Q},\mathbf{A},\mathbf{b},\mathbf{c})$ as:

$$\begin{aligned} \mathbf{Q} &= \nabla_\theta u_{\theta_t}(\mathbf{X})^\top\nabla_\theta u_{\theta_t}(\mathbf{X}), \\ \mathbf{A} &= H[\nabla_\theta u_{\theta_t}\Delta t](\mathbf{X}_H), \\ \mathbf{c} &= -F[u_{\theta_t}](\mathbf{X})^\top\nabla_\theta u_{\theta_t}(\mathbf{X}), \\ \mathbf{b} &= -H[u_{\theta_t}](\mathbf{X}_H). \end{aligned} \tag{5}$$

Adaptive primal-dual hybrid gradient (PDHG) Algorithm for QP. The QP problem in Eq. 3 can be solved by various classic algorithms, e.g. matrix-factorization-based methods including interior-point and simplex, and first-order methods based on matrix-vector production including PDHG and ADMM. Considering the high-dimensional nature, we choose PDHG for its simplicity and scalability. It converts the QP problem into a saddle-point problem:

$$\min_{\mathbf{x}\in[\mathbf{l},\mathbf{u}]}\max_{\mathbf{y}\geq 0} \quad \frac{1}{2}\mathbf{x}^\top\mathbf{Q}\mathbf{x}+\mathbf{c}^\top\mathbf{x}-\mathbf{y}^\top(\mathbf{A}\mathbf{x}-\mathbf{b}), \tag{6}$$

Algorithm 1 A-PDHG embodiment for solving QP in Eq. 3

Input: Step-size $\{(\tau_k, \sigma_k, \beta_k, \gamma_k)\}$, initial points $(\mathbf{x}^0, \mathbf{y}^0)$, QP instance $(\mathbf{Q}, \mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{l}, \mathbf{u})$
Initialize $\bar{\mathbf{x}}^0 = x^0$
for $k = 0, 1, 2, \ldots$ **do**
  $\mathbf{x}_{md}^k = (1-\beta_k)\bar{\mathbf{x}}^k + \beta_k \mathbf{x}^k$
  $\mathbf{x}^{k+1} = \mathrm{Proj}^{\mathbf{x}}_{[\mathbf{l},\mathbf{u}]}\{\mathbf{x}^k - \tau_k(\mathbf{Q}\mathbf{x}_{md}^k + c - A^\top \mathbf{y}^k)\}$
  $\mathbf{y}_{md}^{k+1} = \mathbf{y}^k + \sigma_k(\mathbf{b} - A(\gamma_k(\mathbf{x}^{k+1} - \mathbf{x}^k) + \mathbf{x}^{k+1}))$
  $\mathbf{y}^{k+1} = \mathrm{Proj}^{y}_{\mathbb{R}_+}\{\mathbf{y}_{md}^{k+1}\}$
  $\bar{\mathbf{x}}^{k+1} = (1-\beta_k)\bar{\mathbf{x}}^k + \beta_k \mathbf{x}^{k+1}$
  if$(\mathbf{x}^k, \mathbf{y}^k)$ converges **then break**
Output: Solution $\mathbf{x}^k$;

and iteratively updates the primal and dual variables $\mathbf{x}, \mathbf{y}$ from initial guesses, leveraging gradient information. Among the variants of PDHG, the momentum-accelerated PDHG (A-PDHG) (Lu & Yang, 2023) is chosen for its fast convergence, as summarized in Algorithm 1.

The Advantages of A-PDHG. Prior high–dim neural operators cast parameter evolution as an *unconstrained* least-squares fit or treat it as a black-box regression, making it impossible to encode hard safety or no-arbitrage rules. By rewriting the evolution step as a quadratic programming (QP) we can *natively* embed linear inequality information and thus extend to a broader spectrum of constrained PDEs.

A vanilla A-PDHG solver typically needs 100+ primal–dual iterations on our largest QP instances—prohibitively expensive during back-propagation. HEAP truncates the algorithm to $K = 3$ steps and *learns* step sizes, momentum and expansion weights from data, retaining algorithmic priors while reducing compute time by two orders of magnitude. Table 6 confirms that $K \leq 3$ already matches, or surpasses, the accuracy of the full A-PDHG loop.

3.2. Unroll A-PDHG into Neural Architecture

Despite the fast theoretical convergence rate of A-PDHG, it still suffers long-tail convergence in practice, partially due to the difficulty in tuning the hyperparameters. In our PDE evolution operator settings, the long iterations lead to slow and unstable backpropagation. To address this issue, we truncate and extend A-PDHG into a hyper-network architecture, termed as Hyper-Extended A-PDHG (HEAP), with the following modifications:

1) Truncate the iterations to a fixed number K. This is motivated by the observation that the A-PDHG algorithm converges quickly in the first few iterations.

2) Extend the computation to a learnable latent space, compensating for the inaccuracy due to truncation. The latent space is parameterized by linear expansion weights $\{W\}$ on intermediate variables.

Algorithm 2 Hyper-Extended A-PDHG (i.e. HEAP)

Input: ansatz parameter θ_t, hypernet $\mathcal{N}_w$, iteration K
Assemble QP instance $(\mathbf{Q}, \mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{l}, \mathbf{u})$ from Eq. 5
Estimate initial points $(\mathbf{X}^0, \mathbf{Y}^0)$, step-sizes and expansion weights $\{\mathbf{W}\}$ by $\mathcal{N}_w(\theta_t)$
Encode $\mathbf{X}^0 = \mathbf{x}^0(\mathbf{1} + \mathbf{W}_x^e)^\top$, $\mathbf{Y}^0 = \mathbf{y}^0(\mathbf{1} + \mathbf{W}_y^e)^\top$
Initialize $\bar{\mathbf{X}}^0 = \mathbf{X}^0$
for $k = 0, 1, 2, \ldots, K-1$ **do**
  $\mathbf{X}_{md}^k = (1-\beta_k)\bar{\mathbf{X}}^k + \beta_k \mathbf{X}^k$
  $\mathbf{X}_{md}^k = \mathbf{X}^k - \tau_k(\mathbf{Q}\mathbf{X}_{md}^k \mathbf{W}_{md}^k + \mathbf{c} - \mathbf{A}^\top \mathbf{Y}^k \mathbf{W}_y^k)$
  $\mathbf{X}^{k+1} = \mathrm{Clamp}_{[\mathbf{l},\mathbf{u}]}(\mathbf{X}_{md}^k)$
  $\mathbf{Y}_{md}^{k+1} = \mathbf{Y}^k + \sigma_k(b - \mathbf{A}(\gamma_k(\mathbf{X}^{k+1} - \mathbf{X}^k) + \mathbf{X}^{k+1})\mathbf{W}_x^k)$
  $\mathbf{Y}^{k+1} = \mathrm{ReLU}(\mathbf{Y}_{md}^{k+1})$
  $\bar{\mathbf{X}}^{k+1} = (1-\beta_k)\bar{\mathbf{X}}^k + \beta_k \mathbf{X}^{k+1}$
Decode $\mathbf{x}^K = (\mathbf{X}^K - \mathbf{x}^0 \mathbf{W}_x^e)\mathbf{1}^\top$
Output: Parameter evolution estimation $\dot{\theta}_t = \mathbf{x}^K$

3) Estimate the initial points $(\mathbf{x}^0, \mathbf{y}^0)$, step-sizes and expansion weights $\{W\}$ by a hypernetwork $\mathcal{N}_w(\theta_t)$. The hypernetwork weights w are trainable.

4) Convert the projection operators to differentiable functions, e.g. ReLU, Clamp, to enable end-to-end training.

The HEAP is shown in Algorithm 2, with the modifications marked as blue, and the workflow is illustrated in Fig. 1.

Training and Testing. The training loss is the accumulated residual r_t along the time horizon, whose increment $\dot{r}_t$ is the weighted sum of the PDE residual $\dot{r}_{t,\text{pde}}$, the physical constraint violation $\dot{r}_{t,\text{pc}}$, and the numerical constraints $\dot{r}_{t,\text{nc}}$.

$$\begin{aligned}
\dot{r}_{t,\text{pde}} &= \left\| \dot{\theta}_t^\top \mathbf{Q} \dot{\theta}_t + \mathbf{c}^\top \dot{\theta}_t \right\|^2 \\
\dot{r}_{t,\text{pc}} &= \left\| \min(\mathbf{A}\dot{\theta}_t - \mathbf{b}, 0) \right\|^2 / \left\| \mathbf{A}\dot{\theta}_t \right\|^2 \\
\dot{r}_{t,\text{nc}} &= \left\| \min(\dot{\theta}_t - \mathbf{l}, \mathbf{u} - \dot{\theta}_t, 0) \right\|^2 / \left\| \dot{\theta}_t \right\|^2 \\
\dot{r}_t &= \lambda_{\text{pde}} \dot{r}_{t,\text{pde}} + \lambda_{\text{pc}} \dot{r}_{t,\text{pc}} + \lambda_{\text{nc}} \dot{r}_{t,\text{nc}},
\end{aligned} \tag{7}$$

where the $\{\lambda\}$ are tunable scalar weights. The forward pass is the NeuralODE (Chen et al., 2018) integration of both the residual r_t and the ansatz parameter θ_t:

$$\begin{aligned}
r_T &= \int_0^T r_t \, \mathrm{d}t \\
\theta_T &= \theta_0 + \int_0^T V_w^{HEAP}(\theta_t) \, \mathrm{d}t
\end{aligned} \tag{8}$$

All operations in the HEAP model are differentiable, and the gradients are computed by backpropagation via the adjoint method (Chen et al., 2018). The adjoint method is

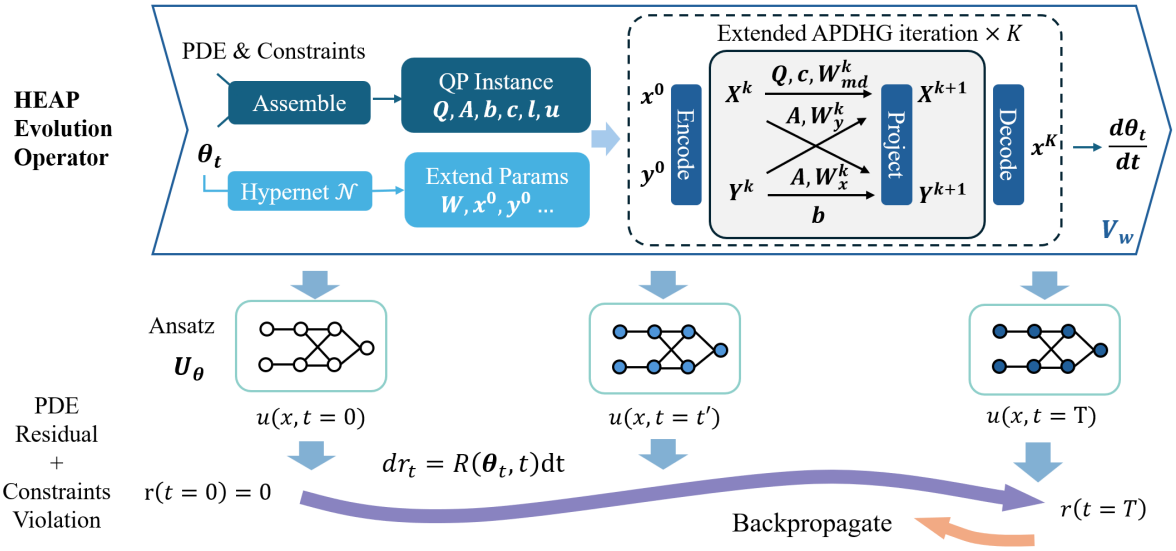

Figure 1. The PDE evolution is embedded into the ansatz parameter space, approximated as a QP operator and solved by the HEAP. Its operator is unrolled from the A-PDHG algorithm, with the iterations truncated and extended to a learnable latent space. The hypernetwork $\mathcal{N}_w$ estimates the initial points, step sizes, and expansion weights. The projection operators are replaced by differentiable functions.

an efficient technique for computing gradients by solving backward equations, which propagate gradient information from the output to the input.

In inference, first the initial parameters θ_0 are obtained by fitting the ansatz with the initial function. Then θ_0 is fed to the trained HEAP model to get the solution function. The test metric is the PDE relative residual at the final time T:

$$\text{L2RE}_{\text{pde}} = \|u_{\theta_T} - u_{ref}\|/\|u_{ref}\|. \tag{9}$$

Intuitive view. *A-PDHG algorithm.* Adaptive PDHG alternates a primal descent on θ (reducing mismatch) with a dual ascent on the Lagrange multipliers y (penalising violations). Each update is projected onto the feasible set.

HEAP network. HEAP unrolls a *fixed* number K of A-PDHG iterations into a feed-forward network. Primal and dual variables are lifted from vectors to matrices—effectively solving K QP steps in parallel—and the lifting weights are produced by a hyper-network that perceives the current PDE state. The entire construct is end-to-end differentiable.

3.3. Theoretical Analysis

We provide our theoretical study as follows.

Alignment with A-PDHG Algorithm. Firstly, we prove HEAP's alignment with A-PDHG, i.e. there exists a HEAP that could replicate the primal-dual sequence generated by A-PDHG. A stricter expression is provided as follows, and the proof of this theorem is given in Appendix A.

Theorem 3.1. ***[Existence of HEAP]*** *Given any QP instance $\mathcal{M} = (Q, A, b, c, l, u)$, HyperNet $\mathcal{N}$, and its corresponding primal-dual sequence $(x_k, y_k)_{k\le K}$ generated by the PDQP algorithm within K iterations, there exists a K-layer HEAP with parameter assignment Θ_{HEAP} that can output the same iterative solution sequence.*

Approximation Capability. Secondly, we verify the approximation capacity of HEAP. The proof of this theorem is given in Appendix A.

Theorem 3.2. ***[Linear Convergence]*** *Given an approximation error bound ϵ, there exists a K-layer HEAP with $\mathcal{O}(\frac{1}{\epsilon})$ neurons that exhibits linear convergence, i.e. for any QP instance $\mathcal{M} = (Q, A, b, c, l, u)$ and $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m_{\ge 0}$ satisfying $l \le x \le u$, it follows that*

$$L(X^K, y; \mathcal{M}) - L(x, Y^K; \mathcal{M}) < \epsilon \tag{10}$$

4. Experiments

All experiments were conducted on a machine with 1TB memory, 144 cores Intel Xeon Platinum 8352V CPU, and 8 GPUs (NV GeForce RTX 4090).

4.1. PDEs Selection

We benchmark HEAP on *four* prototype equations, each tested in both an unconstrained form (suffix `-UC`) and a constrained form (suffix `-C`). For `-C` cases we enforce inequality constraints that must hold *everywhere* in

space–time; for -UC we drop those constraints but keep the same governing PDE. What's more, the numerical constraints which are applied in both -C and -UC cases are the bound $\mathbf{l}, \mathbf{u}$ of parameters θ_t, which is chosen as $\mathbf{l} = -\mathbf{1}, \mathbf{u} = \mathbf{1}$. Unless noted otherwise, all experiments use spatial dimension $d \in \{5, 10, 15, 20\}$.

Heat Equation.

$$\begin{aligned} \partial_t u &= \Delta_x u, \quad x \in [-1,1]^d,\ t \in (0,T], \\ u(x,0) &= g_{\text{heat}}(x) = \alpha \prod_{i=1}^{d} \sin(\pi x_i), \\ \partial_t u &\le \lambda_{\text{heat}} \quad \text{(only for Heat-C)}. \end{aligned} \tag{17}$$

where $\alpha \in [0,1]$ is a tunable parameter.

Burgers Equation.

$$\begin{aligned} \partial_t u &= \tfrac{1}{2}\sigma^2 \Delta_x u + \Big(u - \tfrac{2+d}{2d}\Big) \sum_{i=1}^{d} \partial_{x_i} u, \\ u(x,0) &= g_{\text{burg}}(x) = 1 - \frac{1}{1 + \exp\big(k \cdot t + \frac{1}{d}\sum_i x_i\big)}, \\ \partial_t u &\le \lambda_{\text{burg}} \quad \text{(only for Burgers-C)}. \end{aligned} \tag{18}$$

where $k \in [0.8, 1.3]$ is the the adjustable coefficient.

Reaction–Diffusion (RD) Equation.

$$\begin{aligned} \partial_t u &= \tfrac{1}{2}\Delta_x u + \min\{1, f^2(t,x,u)\}, \\ f(t,x,u) &= u - \kappa - 1 - \sin\Big(\lambda \sum_{i=1}^{d} x_i\Big) e^{\frac{\lambda^2 d}{2}(t-T)}, \\ u(x,0) &= g_{\text{rd}}(x) = 1 + \kappa + \sin\Big(\lambda \sum_{i=1}^{d} x_i\Big), \\ \partial_t u &\le \lambda_{\text{rd}} \quad \text{(only for RD-C)}. \end{aligned} \tag{19}$$

where $\lambda = 1/\sqrt{d}$ is a constant representing the wave frequency, $\kappa \in [0.5, 1.5]$ is the tunable parameters.

Black–Scholes (BS) Equation.

$$\begin{aligned} \partial_t u &= \sum_{i=1}^{d} \mu\, x_i\, \partial_{x_i} u + \frac{1}{2}\sigma^2 \sum_{i=1}^{d} x_i^2\, \partial_{x_i}^2 u \\ &\quad - \big[(1-\delta)\, Q(u) + r\big] u, \\ Q(u) &= \text{ReLU}\Big(\text{ReLU}(u - v_h)\frac{\gamma_h - \gamma_l}{v_h - v_l} + \gamma_h - \gamma_l\Big) + \gamma_l, \\ u(x,0) &= g_{\text{bs}}(x) = k \min_i x_i, \\ \partial_t u &\ge r\, u \qquad \text{(T1, only for BS-C)}, \\ \partial_t u &\ge r\, u - \lambda_{\text{bs}} \sum_{i=1}^{d} \partial_{x_i}^2 u \quad \text{(T2, only for BS-C)}. \end{aligned} \tag{20}$$

Table 1. Hyper-parameters for CSO on Burgers; HEAP uses identical values. T represents the time range of the equation, **steps** denotes the number of iterations of the equation in time.

dim	T	steps	input_dim	dropout	learning_rate
5	0.2	10	1921	0.30	5×10^{-5}
10	0.2	10	2121	0.30	5×10^{-5}
15	0.2	20	2321	0.30	2×10^{-4}
20	0.2	20	2521	0.30	3×10^{-4}

Parameters μ, σ, r denote drift, volatility and the risk-free rate; $\delta, v_h, v_l, \gamma_h, \gamma_l$ are calibration constants. $k \in [0.5, 1.5]$ is the the adjustable coefficient. Constraints (T1)–(T2) encode standard no-arbitrage bounds on the option's theta.

4.2. Experiments Implementation Details

Setup. The extended dimension of HEAP is 20; the iteration is 3, and the hypernetwork $\mathcal{N}_w$ is a 5-layer ResNet with 1000 hidden units. The HEAP is trained with Adam optimizer, default batch size 64, and 100 epochs. E.g., Table 1 summarises the settings used for CSO on our Burgers type equation experiments; HEAP employs the same configuration. The ODE is integrated with the Runge-Kutta (Butcher, 1996) with a time step $1/10$ of the time horizon.

Sampling of initial parameters. The training set consists of 150K randomly generated θ from normal distribution, and the test set is 250 randomly generated θ from another normal distribution with slightly different mean and variance. Unless otherwise noted, the initial ansatz parameters θ_0 are drawn i.i.d. from a Gaussian distribution with mean 0 and standard deviation $\sqrt{0.5}$ for training, and from a wider $\mathcal{N}(0,1)$ for test; this encourages robustness to distribution shift while keeping the training residual numerically stable.

Composite loss weights. The accumulated residual $\dot{r}_t$ (see Eq. (7)) is a weighted sum of three terms: (i) PDE residual, (ii) physical constraint violation, and (iii) numerical box-constraint violation. To balance their magnitudes during optimisation we set $\lambda_{pde} = 10^{-2}$, $\lambda_{pc} = 1$, and $\lambda_{nc} = 10$, respectively, for all experiments. The same coefficients are applied to the baseline CSO to ensure fair comparison.

4.3. Peer Methods and Protocols

To our best knowledge, only a handful of neural–operator models can handle high–dimensional PDEs, and none of them explicitly target strong inequality constraints. We thus adopt the recently proposed *Control-based Solution Operator* (CSO) (Gaby & Ye, 2024) as the SOTA baseline. CSO parametrises the parameter trajectory $\dot{\theta}_t$ with a black–box network; for fair comparison we reuse HEAP's hypernetwork backbone, identical optimiser settings, batch size and training horizon across all tasks.

Table 2. PDE residual $\mathrm{L2RE}_{\mathrm{pde}}$ on unconstrained equations comparison between the strong and recently proposed SOTA baseline CSO (Gaby & Ye, 2024) and our proposed HEAP on various PDEs.

Eqn	Heat			
Dim	d=5	d=10	d=15	d=20
CSO	0.0222	0.0218	0.0218	0.0273
HEAP	**0.0041**	**0.0068**	**0.0106**	**0.0141**
Eqn	Burgers type			
CSO	0.0028	0.0058	0.0093	0.0095
HEAP	**0.0020**	**0.0032**	**0.0079**	**0.0089**
Eqn	Reaction Diffusion			
CSO	0.0028	0.0024	0.0076	0.0021
HEAP	**0.0021**	**0.0016**	**0.0024**	**0.0020**
Eqn	Black-Scholes Equation			
CSO	0.0329	0.0439	0.0391	0.0494
HEAP	**0.0270**	**0.0308**	**0.0340**	**0.0433**

Metrics with ground-truth reference.
For those PDEs whose exact or high–fidelity reference solutions are available (*Heat-UC/C*, *Burgers-UC*, *RD-UC*, *BS-UC*), we report the L2 relative error of the PDE residual defined in Eq. (9). References are analytical for Heat, and produced by a high-resolution *DeepBSDE* solver for Burgers, RD and BS.

Metrics without ground-truth reference.
For *Burgers-C*, *RD-C* and *BS-C* the reference solution is unknown; we therefore adopt the accumulated residual $\dot{r}_t$ (see Eq. (7)), which jointly measures the physics error and constraint mismatch along the trajectory.

Constraint satisfaction.
In every constrained task we additionally report the normalised terminal-time physical constraint violation $\dot{r}_{t,\mathrm{pc}}$ (Eq. (7)), irrespective of the availability of ground truth.

4.4. Results and Discussion

Results for Unconstrained Equations. The PDE residual on unconstrained equations (*Heat-UC*, *Burgers-UC*, *RD-UC*, *BS-UC*) are compared in Table 2. The L2RE of PDE of HEAP is consistently lower than that of CSO, indicating better accuracy and generalization.

Results for Constrained Equations. The PDE residual on constrained equations (*Heat-C*, *Burgers-C*, *RD-C*, *BS-C-T1*, *BS-C-T2*) are compared in Table 3. As mentioned in 4.1.1, we set some reasonable constraints for each type of equation, where the Black-Scholes equation contains two relatively more complex no-arbitrage constraints. The L2RE of PDE of HEAP is consistently lower than that of CSO, indicating better accuracy and generalization. By comparison with Table 2, it can be found that under constrained conditions, the improvement of the HEAP to CSO is more significant, demonstrating the effectiveness of HEAP in solving high-dimensional PDEs with constraints.

Table 3. PDE residual $\mathrm{L2RE}_{\mathrm{pde}}$ (Only for Heat-C) and accumulated PDE residual $\dot{r}_t$ for (Others) on constrained equations comparison between the strong and recently proposed SOTA baseline CSO (Gaby & Ye, 2024) and our proposed HEAP on various PDEs.

Eqn	Constrained Heat			
Dim	d=5	d=10	d=15	d=20
CSO	0.0030	0.0187	0.0136	0.0407
HEAP	**0.0025**	**0.0044**	**0.0092**	**0.0169**
Eqn	Constrained Burgers Type			
CSO	0.9798	0.9125	0.9764	0.9529
HEAP	**0.7377**	**0.6281**	**0.6926**	**0.8778**
Eqn	Constrained Reaction Diffusion			
CSO	0.3560	0.4746	0.5687	0.5102
HEAP	**0.3560**	**0.3768**	**0.4730**	**0.3900**
Eqn	Constrained Black-Scholes Type 1			
CSO	0.9462	0.8649	0.9145	1.000
HEAP	**0.7022**	**0.5667**	**0.5232**	**0.4756**
Eqn	Constrained Black-Scholes Type 2			
CSO	0.9570	0.8625	0.9260	1.0000
HEAP	**0.6046**	**0.4241**	**0.4617**	**0.4670**

The normalized terminal-time physical constraint violation $\dot{r}_{t,\mathrm{pc}}$ on constrained equations are also compared in Table 4. Experimental results show that the Heap performs excellent in satisfying the physical constraints of the equation. Especially in the Black-Scholes equation with more complex constraints, Heap significantly reduces the physical constraint violation compared with the CSO, showing the potential of Heap to solve complex linear constraints.

To intuitively show the advantages of the Heap over the CSO baseline, we visualized the solution and resisual of the constrained 10-dimensional Black-Scholes equation on Fig.2. We changed two of the dimensions and set the other dimensions to 0, and performed a grid solution on a $[0, 1]^2$ plane. The image indicate that the accumulated PDE residual of the Heap is significantly smaller than that of CSO.

Time and Space Usage. The table 5 shows the performance indicators of Heap and CSO methods for Dim=5, 10, 15, 20, 50, 100, include time consumption, CPU memory usage(CMem), and GPU memory usage(GMem). Without any optimization applied, the HEAP costs 1.6x GPU memory and about 2x training/inference time compared to the baseline CSO. The time and memory usage are nearly linearly increasing with the dimension, thus HEAP is scalable to

Table 4. physical constraint violation $\mathrm{L2RE}_{\mathrm{con}}$ comparison between CSO and HEAP on constrained PDEs. The **best** is marked.

Eqn	Constrained Heat			
Dim	d=5	d=10	d=15	d=20
CSO	0.2944	0.2843	**0.0080**	1.58×10^{-4}
HEAP	**0.2540**	**0.2234**	0.0168	3.21×10^{-5}
Eqn	Constrained Burgers Type			
CSO	0.4544	0.5606	**0.4335**	0.4098
HEAP	**0.4224**	**0.4603**	0.5948	**0.3345**
Eqn	Constrained Reaction Diffusion			
CSO	0.2556	0.7593	0.6725	0.6936
HEAP	**0.2555**	**0.2585**	**0.2359**	**0.1794**
Eqn	Constrained Black-Scholes Type 1			
CSO	0.6368	0.9386	0.6837	0.694
HEAP	**0.2843**	**0.1932**	**0.3310**	**0.2231**
Eqn	Constrained Black-Scholes Type 2			
CSO	0.7466	0.7919	0.6546	0.6936
HEAP	**0.5595**	**0.3865**	**0.0719**	**0.0165**

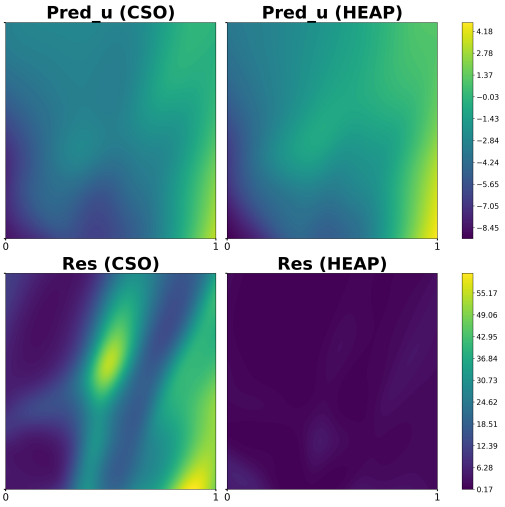

Figure 2. The predict solution (Pred_u) and accumulated PDE residual (Res) of Heap and CSO in Black-Scholes Equation (dim-10) with constraint type 2.

higher dimensions in terms of computation.

Ablation Experiment for the hypernetwork. The number of unrolling layers and extended dimension in the hypernetwork are the core hyperparameters of the Heap method. We show the results of the ablation experiment on these two hyperparameters in Table 6 (taking the 15-dimensional Black-Scholes equation as an example), whose metrics value is also the accumulated residual $\dot{r}_t$ (see Eq. (7)). The results show that both unrolling layers = 1,2,3,4 and extended dimension = 5,10,20 in the hypernetwork have a significant impact on final accuracy and constraint satisfaction. However, $K = 3$ is a relatively robust choice across different PDE families.

Table 5. Comparison of runtime (sec.) and memory usage (MB) for CSO and HEAP on Burgers-type equations. The test time is evaluated with 1 batch, and the training time is calculated with 3 and 156 batches (156 batches is approximately correspond to 1 epoch). The batchsize is set as 64. The symbol * indicates that training was not completed due to GPU memory overflow.

Dim	**Method**	**CMem**	**GMem**	T_1^{test}	T_3^{train}	T_{156}^{train}
5	CSO	4.47	12336	0.90	24.12	596
5	HEAP	4.50	15865	2.94	67.48	1071
10	CSO	4.21	14157	1.21	32.06	699
10	HEAP	4.15	18696	3.52	79.63	1136
15	CSO	4.44	16075	1.97	42.02	902
15	HEAP	4.57	21927	4.31	97.87	1668
20	CSO	4.15	18085	2.31	53.44	1119
20	HEAP	4.22	25405	4.92	112.83	1896
50	CSO	4.39	32166	4.85	109.90	/
50	HEAP	4.50	51654	11.19	198.46	/
100*	CSO	4.05	71958	11.54	/	/
100*	HEAP	4.29	80000*	25.46	/	/

Table 6. Ablation study on the number of unrolling layers and extened dims in HEAP on BS equation with constraint type 1. The PDE residuals are normalized by a constant scale.

		Unrolling Layers			
Extended Dim	Residual	1	2	3	4
5	PDE	0.9630	0.5205	1.7511	4.5298
	Con	0.5084	0.1163	0.4339	0.3460
10	PDE	0.8557	0.5246	0.5857	1.5917
	Con	0.5869	0.3310	0.7472	0.6787
20	PDE	0.9181	0.4298	0.5283	1.5857
	Con	0.3455	0.1203	0.2708	0.6474

5. Conclusion

We propose a novel neural operator for Constrained High-dim PDEs and devise the HEAP architecture for end-to-end training. We theoretically prove the approximation capability and provide empirical results on a variety of high-dim PDEs to demonstrate that it outperforms existing state-of-the-art neural operator models and shows better constraint satisfaction and generalization.

Impact Statement

This paper presents the work that aims to advance the field of Machine Learning for Science (ML4Sci). There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Alet, F., Jeewajee, A. K., Villalonga, M. B., Rodriguez, A., Lozano-Perez, T., and Kaelbling, L. Graph element networks: adaptive, structured computation and memory. In *International Conference on Machine Learning*, pp. 212–222. PMLR, 2019.

Bellman, R. Dynamic programming. *science*, 153(3731): 34–37, 1966.

Blechschmidt, J. and Ernst, O. G. Three ways to solve partial differential equations with neural networks—a review. *GAMM-Mitteilungen*, 44(2):e202100006, 2021.

Bryutkin, A., Huang, J., Deng, Z., Yang, G., Schönlieb, C.-B., and Aviles-Rivero, A. Hamlet: Graph transformer neural operator for partial differential equations. *arXiv preprint arXiv:2402.03541*, 2024.

Butcher, J. C. A history of runge-kutta methods. *Applied numerical mathematics*, 20(3):247–260, 1996.

Chalapathi, N., Du, Y., and Krishnapriyan, A. Scaling physics-informed hard constraints with mixture-of-experts. *arXiv preprint arXiv:2402.13412*, 2024.

Chambolle, A. and Pock, T. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1):253–287, 2016.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Chen, T. and Chen, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

De Ryck, T., Mishra, S., and Molinaro, R. wpinns: Weak physics informed neural networks for approximating entropy solutions of hyperbolic conservation laws. *SIAM Journal on Numerical Analysis*, 62(2):811–841, 2024.

Du, Y., Chalapathi, N., and Krishnapriyan, A. Neural spectral methods: Self-supervised learning in the spectral domain. *arXiv preprint arXiv:2312.05225*, 2023.

Fefferman, C. L. Existence and smoothness of the navier–stokes equation. In Carlson, J., Jaffe, A., and Wiles, A. (eds.), *The Millennium Prize Problems*, pp. 57–67. Clay Mathematics Institute, 2006.

Feng, M., Huang, Y., Liu, Y., Liao, W., Liu, Y., and Yan, J. Singer: Stochastic network graph evolving operator for high dimensional pdes. *ICLR*, 2025.

Folland, G. B. *Introduction to partial differential equations*. Princeton university press, 2020.

Gaby, N. and Ye, X. Approximation of solution operators for high-dimensional pdes. *arXiv preprint arXiv:2401.10385*, 2024.

Gaby, N., Ye, X., and Zhou, H. Neural control of parametric solutions for high-dimensional evolution pdes. *SIAM Journal on Scientific Computing*, 46(2):C155–C185, 2024.

Han, J., Jentzen, A., et al. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in mathematics and statistics*, 5 (4):349–380, 2017.

Han, J., Jentzen, A., and E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34): 8505–8510, 2018.

Horie, M. and Mitsume, N. Physics-embedded neural networks: Graph neural pde solvers with mixed boundary conditions. *Advances in Neural Information Processing Systems*, 35:23218–23229, 2022.

Hoshisashi, K., Phelan, C. E., and Barucca, P. Physics-informed neural networks for derivative-constrained pdes. In *Proceedings of the ICML 2024 Workshop on AI for Science*, 2024. to appear.

Hu, Z., Shukla, K., Karniadakis, G. E., and Kawaguchi, K. Tackling the curse of dimensionality with physics-informed neural networks. *Neural Networks*, 176:106369, 2024.

Huebner, K. H., Dewhirst, D. L., Smith, D. E., and Byrom, T. G. *The finite element method for engineers*. John Wiley & Sons, 2001.

Hull, J. C. and Basu, S. *Options, Futures, and Other Derivatives*. Pearson Education, Boston, 9 edition, 2016.

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

Lagaris, I. E., Likas, A., and Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5): 987–1000, 1998.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A., et al. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2020.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science*, 1(3):1–27, 2024.

List, B., Chen, L.-W., and Thuerey, N. Learned turbulence modelling with differentiable fluid solvers: physics-based loss functions and optimisation horizons. *Journal of Fluid Mechanics*, 949:A25, 2022.

Lötzsch, W., Ohler, S., and Otterbach, J. S. Learning the solution operator of boundary value problems using graph neural networks. *arXiv preprint arXiv:2206.14092*, 2022.

Lu, H. and Yang, J. A practical and optimal first-order method for large-scale convex quadratic programming. *arXiv preprint arXiv:2311.07710*, 2023.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021a.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., and Johnson, S. G. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021b.

McClenny, L. and Braga-Neto, U. Self-adaptive physics-informed neural networks using a soft attention mechanism. *arXiv preprint arXiv:2009.04544*, 2020.

Moro, V. and Chamon, L. F. O. Solving differential equations with constrained learning. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. to appear.

Négiar, G., Mahoney, M. W., and Krishnapriyan, A. S. Learning differentiable solvers for systems with hard constraints. *arXiv preprint arXiv:2207.08675*, 2022.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. 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.

Richter-Powell, J., Lipman, Y., and Chen, R. T. Neural conservation laws: A divergence-free perspective. *Advances in Neural Information Processing Systems*, 35: 38075–38088, 2022.

Saad, N., Gupta, G., Alizadeh, S., and Maddix, D. C. Guiding continuous operator learning through physics-based boundary constraints. *arXiv preprint arXiv:2212.07477*, 2022.

Saleh, E., Ghaffari, S., Bretl, T., Olson, L., and West, M. Learning from integral losses in physics informed neural networks. *arXiv preprint arXiv:2305.17387*, 2023.

Shang, Y., Wang, F., and Sun, J. Randomized neural network with petrov–galerkin methods for solving linear and nonlinear partial differential equations. *Communications in Nonlinear Science and Numerical Simulation*, 127: 107518, 2023.

Shi, Z., Hu, Z., Lin, M., and Kawaguchi, K. Stochastic taylor derivative estimator: Efficient amortization for arbitrary differential operators. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Strauss, W. A. *Partial differential equations: An introduction*. John Wiley & Sons, 2007.

Wang, H., Li, J., Dwivedi, A., Hara, K., and Wu, T. Beno: Boundary-embedded neural operators for elliptic pdes. *arXiv preprint arXiv:2401.09323*, 2024.

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

Wang, Y., Jin, P., and Xie, H. Tensor neural network and its numerical integration. *arXiv preprint arXiv:2207.02754*, 2022.

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

Zang, Y., Bao, G., Ye, X., and Zhou, H. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.

A. Theorem Proof

Proof of Theorem 3.1. We use mathematical induction.

Base Step. At iteration $k = 0$, both APDQP and HEAP are initialized by hypernet $\mathcal{N}_w$, and the base case holds trivially.

Inductive Step. Assume that for some $k \geq 0$, $X^k = x^k$ and $Y^k = y^k$. We now show that $X^{k+1} = x^{k+1}$ and $Y^{k+1} = y^{k+1}$ for a suitable choice of parameters in Θ_{HEAP}.

For the APDQP update, we know:

$$\begin{aligned} x^{k+1} &= \text{Proj}^x_{[l,u]}(x^k - \tau_k(Qx^k_{md} + c - A^\top y^k)) \\ y^{k+1} &= \text{Proj}^y(y^k + \sigma_k(b - A(\gamma_k(x^{k+1} - x^k) + x^{k+1}))) \end{aligned} \tag{11}$$

The corresponding HEAP updates are:

$$\begin{aligned} X^{k+1} &= \text{Clamp}_{[l,u]}(X^k - \tau_k(QX^k_{md}W^k_x + c - A^\top Y^k W^k_y)) \\ Y^{k+1} &= \text{ReLU}(Y^k + \sigma_k(b - A(\gamma_k(X^{k+1} - X^k) \\ &\quad + X^{k+1})W^k_x)) \end{aligned} \tag{12}$$

By setting $W^k_x = W^k_y = I$, and ensuring that τ_k, σ_k and γ_k in HEAP match the corresponding step sizes in APDQP, it can be verified that:

$$X^{k+1} = x^{k+1} \quad \text{and} \quad Y^{k+1} = y^{k+1} \tag{13}$$

Thus, by induction, HEAP can exactly replicate the primal-dual sequence generated by the APDQP algorithm, indicating HEAP's expressivity. □

Proof of Theorem 3.2. Firstly, we introduce a proposition that describes A-PDHG's convergence rate.

Proposition A.1. *(Chambolle & Pock, 2016) Let $(x^k, y^k)_{k\geq 0}$ be the primal-dual variables generated by the PDHG algorithm for the QP problem $\mathcal{M} = (Q, A, b, c, l, u)$. If the step sizes τ, σ satisfy $\tau\sigma\|A\|_2^2 < 1$, then for any $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$ satisfying $l \leq x \leq u$, the primal-dual gap satisfies*

$$\begin{aligned} &L(\bar{x}^k, y; \mathcal{M}) - L(x, \bar{y}^k; \mathcal{M}) \\ &\leq \frac{1}{2k}\left(\frac{\|x - x^0\|^2}{\tau} + \frac{\|y - y^0\|^2}{\sigma} - (y - y^0)^\top A(x - x^0)\right) \end{aligned} \tag{14}$$

where $\bar{x}^k = \frac{\Sigma_{j=1}^k x^j}{k}$, $\bar{y}^k = \frac{\Sigma_{j=1}^k y^j}{k}$ and $L(x, y; \mathcal{M}) = \frac{1}{2}\mathbf{x}^\top\mathbf{Q}\mathbf{x} + \mathbf{c}^\top\mathbf{x} - \mathbf{y}^\top(\mathbf{A}\mathbf{x} - \mathbf{b})$.

We can conclude from the proposition above that A-PDHG converges linearly under appropriate conditions. Hence, if we set τ, σ to satisfy $\tau\sigma\|A\|_2^2 < 1$ in both APDQP and HEAP, according to Thm. 3.1, the primal-dual series of HEAP exhibit linear convergence, i.e.

$$\begin{aligned} &L(X^K, y; \mathcal{M}) - L(x, Y^K; \mathcal{M}) \\ &= L(\bar{x}^k, y; \mathcal{M}) - L(x, \bar{y}^k; \mathcal{M}) \\ &\leq \frac{\frac{1}{2k}(\frac{\|x - x^0\|^2}{\tau} + \frac{\|y - y^0\|^2}{\sigma} - (y - y^0)^\top A(x - x^0))}{2K(\epsilon)} \end{aligned} \tag{15}$$

Since $\frac{1}{2k}\left(\frac{\|x - x^0\|^2}{\tau} + \frac{\|y - y^0\|^2}{\sigma} - (y - y^0)^\top A(x - x^0)\right)$ is a constant given a pair of (x, y), we can take $K(\epsilon) = [\frac{C}{\epsilon}]+1$ to obtain:

$$L(X^K, y; \mathcal{M}) - L(x, Y^K; \mathcal{M}) \leq \frac{\epsilon}{2} < \epsilon. \tag{16}$$

Because the network width is a fixed number, the number of neurons at each layer is bounded by a constant C_{num}. Thus, the number of neurons in HEAP is bounded by $C_{num}K(\epsilon)$, which is of $\mathcal{O}(\frac{1}{\epsilon})$. □

B. PDE Solutions under Different Constraints

As discussed in Section 2.3, the solution to the multi-asset Black-Scholes equation may not always be unique in practice. Constraints can be employed to obtain the desired solution. In Figure 3, we compare the convergence of the solution under different constraint conditions (using the Black-Scholes equation from Equation 20, with no constraints and with constraint T2 at varying values of λ_{bs}). The comparison reveals that, even after excluding the influence of solution residuals, the equation yields different solutions in certain regions.

C. The Converge of Learnable Step Sizes

As outlined in Algorithm 2, the A-PDHG method adaptively learns the update step sizes for the X and Y layers (τ and η) to ensure effective training. Through experiments in Figture 6, we have verified that the update step sizes are indeed adjusted during training and converge to a fixed value after approximately 60 batches.

D. Training Trajectory

The training trajectory is visualized in Figures 4 and 5, where we record the PDE residuals at training checkpoints. These figures show that HEAP converges faster and more stably than CSO. Our method's advantage in these benchmarks arises from its principled design and theoretical soundness, particularly through the elegant incorporation of constraints into the neural networks. In contrast, CSO incorporates constraints as a penalty term in the objective, which may lead to constraint violations on the test set.

Although our method demonstrates improvements, we acknowledge that constrained high-dimensional PDEs remain

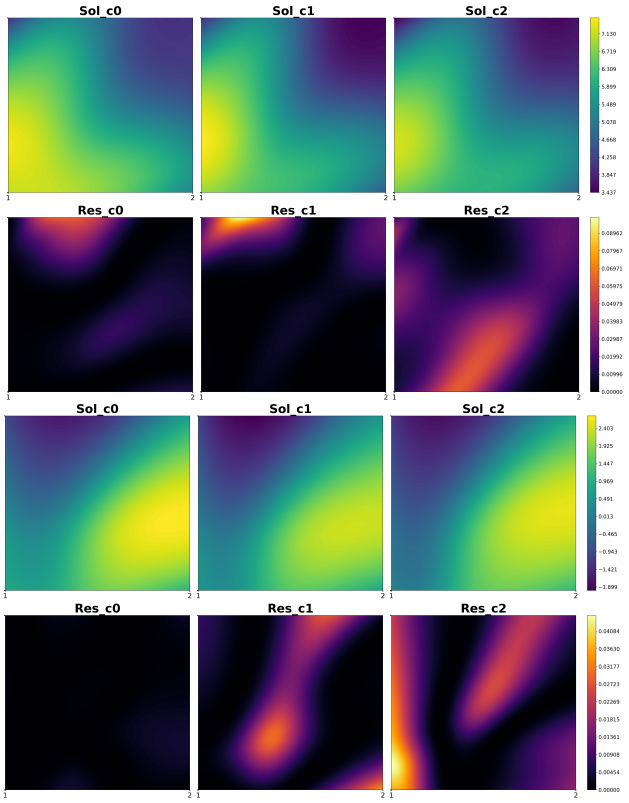

Figure 3. Visualization of predict solution (Pred_u) and accumulated PDE residual (Res) of Heap and CSO in Black-Scholes Equation (dim-10) with no constraints (c0) and two different instances of constraints T2.

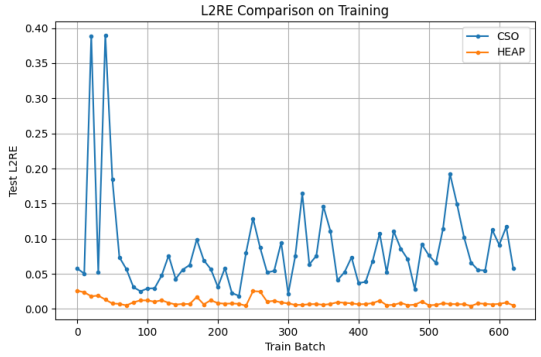

Figure 4. Constrained Heat $\mathrm{L2RE}_{\mathrm{pde}}$ comparison between CSO and our HEAP during the training. Our methods quickly achieve stable (relative) loss over the test set.

challenging due to their inherent nonlinearity. Learning methods still face the risk of overfitting to the training set, which highlights the need for further refinement. It is also worth noting that, to date, we have not extensively fine-tuned the hyperparameters for each specific case.

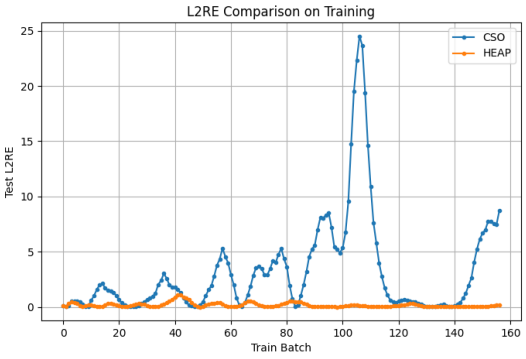

Figure 5. Burgers $\mathrm{L2RE}_{\mathrm{pde}}$ comparison between CSO and our HEAP during the training. The CSO model oscillates and even collapses on the test set.

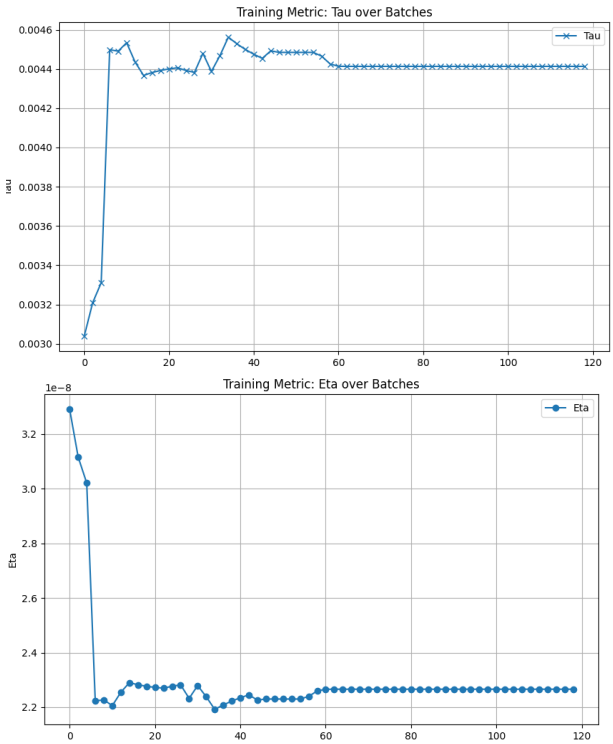

Figure 6. During the training of the 10-dimensional Reaction Diffusion equation, the iteration of the adaptive step size parameter (τ and η) finally converges to a certain value.