LEARNING FAST APPROXIMATE SOLVERS FOR PDE-CONSTRAINED OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Partial differential equation (PDE)-constrained optimization arises in many scientific and engineering domains, such as energy systems, fluid dynamics, and material design. In these problems, the decision variables are tightly coupled with the PDE state variables, and the feasible set is implicitly defined by the governing PDE constraints. This coupling makes these problems computationally demanding, as conventional numerical methods discretize the governing PDE into high-dimensional algebraic systems that must be repeatedly solved throughout the optimization process. To address these challenges, this paper introduces a novel learning-based framework to approximate solutions to PDE-constrained optimization tasks in near real time. The proposed method integrates a dynamic predictor with an optimization surrogate. The dynamic predictor, a time-discrete Neural Operator (Lu et al., 2021), efficiently approximates system trajectories governed by PDE dynamics, while the optimization surrogate leverages proxy optimizer techniques (Kotary et al., 2021) to approximate the associated optimal decisions. This dual-network design enables real-time approximation of optimal strategies while explicitly capturing the coupling between decisions and PDE dynamics. We validate the proposed approach on benchmark PDE-constrained optimization tasks including Burgers' equation, heat equation, and voltage regulation, and demonstrate that it achieves solution quality comparable to state-of-the-art control-based algorithms such as the Direct Method and Model Predictive Control, *while achieving computational speedups of up to four orders of magnitude*.

1 INTRODUCTION

Partial differential equations (PDEs) are central to modeling a wide range of physical and engineering systems, capturing phenomena that evolve jointly over space and time. Applications include fluid transport, which can be modeled with Burgers' equations and are key in fluid mechanics and gas dynamics, thermal processes captured by heat equations, which are often used in financial mathematics and image analysis, and voltage dynamics in power systems. Optimizing decisions in such domains requires solving *PDE-constrained optimization problems*, where control or design objectives must be optimized while ensuring consistency with the underlying governing PDE dynamics.

Despite their importance, solving PDE-constrained optimization can be computationally challenging for many practical applications. Traditional approaches typically rely on finite element and finite difference discretizations combined with adjoint-based optimization (Tröltzsch, 2010). These methods often suffer from high dimensionality, high computational cost, and convergence difficulties when applied to nonlinear dynamics. For example, when applied to the nonlinear Burgers' equations, our experiments show that the runtimes of the adjoint-based and Nonlinear MPC approaches run in approximately 100 s and 870 s; such computational costs hinder their applicability in real-time decision-making tasks.

In various classes of continuous optimization tasks not constrained by PDEs, proxy optimizers (Kotary et al., 2021) have recently emerged as a powerful tool to quickly find approximate solutions. In parallel, researchers have developed a variety of learning-based surrogates (Raissi et al., 2019;

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.

Kovachki et al., 2024; Li et al., 2021; Chen et al., 2018) to accelerate the approximation of system dynamics. Our approach combines elements from both these areas, and consists of a dual-network architecture to tackle PDE-constrained optimization problems where both the state dynamics and the control decisions evolve in space and time.

Contributions. Specifically, the paper makes the following contributions: **(1)** It introduces a novel learning-based method to efficiently approximate solutions of PDE-constrained optimization problems. Our approach, PDE-OP (Optimization Proxy), consists of a dual-network architecture in which a dynamic predictor captures the system dynamics, while a surrogate controller approximates the optimal control actions. **(2)** It introduces a time-discrete Neural Operator tailored for PDE-constrained optimization. **(3)** It demonstrates that the proposed framework achieves decision quality comparable to state-of-the-art numerical solvers, such as Model Predictive Control and the Adjoint-sensitivity method, while achieving orders-of-magnitude reductions in solution time. The approach is validated on a set of widely adopted PDE-constrained optimization tasks, including Burgers', heat, and voltage control. *We believe that these results could boost the adoption of learning-based methods for control problems that require high-fidelity dynamic modeling and real-time decision-making.*

2 Related Work

Here we review existing approaches for optimal control, learning-based constrained optimization, and dynamics modeling. Please refer to Appendix 7.1 for a comprehensive review of these topics.

Classical optimal control methods couple numerical models of system dynamics with iterative optimization. Model Predictive Control (MPC) (Mayne et al., 2000) repeatedly solves finite-horizon problems using predicted trajectories, while adjoint-based PDE-constrained optimization (Tröltzsch, 2010) computes sensitivities through discretized dynamics. Both can be computationally expensive for high-dimensional or nonlinear systems, which limit their adoption in real-time applications.

Proxy optimizers address this computational burden by learning mappings from problem parameters to optimal decisions (Kotary et al., 2021). Many different approaches (Fioretto et al., 2020; Park & Van Hentenryck, 2023) have been explored, with constraint satisfaction remaining a central challenge. In parallel, physics-informed neural networks (Raissi et al., 2019) and operator-learning architectures, including Fourier Neural Operators (Li et al., 2021) and DeepONets (Lu et al., 2021), have enabled efficient approximation of PDE dynamics.

Several approaches combine learned dynamics with control optimization. Surrogate-Based Trajectory Optimization (Hwang et al., 2022) uses operator learning for open-loop control, while DeepONet MPC (de Jong et al., 2025) embeds a learned predictor in a receding-horizon controller. Both these approaches retain numerical optimization at inference. More closely related, Sarkar et al. (2025) combine differentiable predictive control with a pretrained time-integrated DeepONet to learn neural control policies through differentiable surrogate rollouts. Their dynamics model predicts temporal derivatives that are advanced using a numerical time integrator, whereas PDE-OP directly learns a time-discrete state-transition operator that maps the current state and control to the next state. In addition, PDE-OP employs a primal-dual learning formulation to adaptively penalize dynamics and operational constraint violations during controller training.

3 Settings and Goals

This paper focuses on optimization problems constrained by a system of PDEs:

$$\min_{u:Q\to\mathbb{R}} \quad J(u,y) = L(y(T,\cdot)) + \int_0^T\int_\Omega \Phi(u,y)\,dx\,dt + \int_0^T\int_\Omega \|u(t,x)\|_2^2\,dx\,dt \tag{1a}$$

$$\text{s.t.} \quad \partial_t y(t,x) = \mathcal{I}_t(y,u,t,x),\ (t,x)\in Q, \tag{1b}$$

$$\partial_x y(t,x) = \mathcal{I}_x(y,u,t,x),\ (t,x)\in Q, \tag{1c}$$

$$y(t_0,x) = y_{t_0}(x), \quad x\in\Omega, \tag{1d}$$

$$y(t,x) = y_\partial(t,x),\ (t,x)\in[t_0,T]\times\partial\Omega, \tag{1e}$$

$$\boldsymbol{g}(u,y)\le\mathbf{0}, \quad \boldsymbol{h}(u,y)=\mathbf{0},\ (t,x)\in Q. \tag{1f}$$

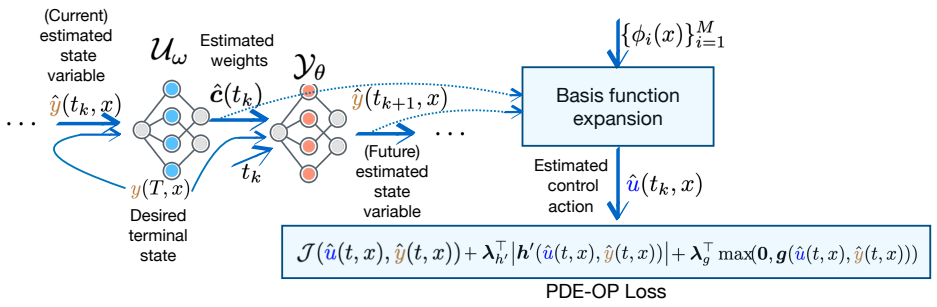

Figure 1: Overview of PDE-OP. At each time instant t_k, PDE-OP uses a dual network architecture consisting of **1)** a proxy optimization model $\mathcal{U}_\omega$ to estimate the basis functions weights $\boldsymbol{c}(t_k)$ and construct the control action $\hat{u}(t_k, x)$, and **2)** a time-discrete neural operator model $\mathcal{Y}_\theta$ to estimate the state-variables $\hat{y}(t_{k+1}, x)$, with the objective function $\mathcal{J}(\hat{u}(t,x), \hat{y}(t,x)) + \boldsymbol{\lambda}_{h'}^\top |\boldsymbol{h}'(\hat{u}(t,x), \hat{y}(t,x))| + \boldsymbol{\lambda}_g^\top \max(\mathbf{0}, \boldsymbol{g}(\hat{u}(t,x), \hat{y}(t,x)))$ capturing the overall loss.

Here $u : Q \to \mathbb{R}$ is the control action, $y : Q \to \mathbb{R}$ is the state, and $Q = [t_0, T] \times \Omega$ is the space–time domain. Equation (1a) defines the objective as the sum of terminal, running, and control-effort costs. Equations (1b)–(1c) define the temporal and spatial dynamics, (1d)-(1e) impose initial and boundary conditions, and (1f) represents additional application-specific requirements.

Challenges. While fundamental for many applications, Problem (1) presents three key challenges:

1. Finding optimal solutions to Problem (1) is computationally challenging.
2. Achieving high-quality approximations of the system dynamics (1b)-(1c) in near real-time, poses the second significant challenge, especially in nonlinear settings.
3. Finally, integrating system dynamics into the optimization framework further increases computational complexity, making traditional numerical methods unsuitable for real-time applications.

4 PDE-Optimization Proxy

To tackle the challenges discussed above, we introduce *PDE-Optimization Proxy* (PDE-OP): a fully differentiable surrogate for PDE-constrained optimization. PDE-OP adopts a dual-network architecture: the first network, $\mathcal{U}_\omega$, approximates the optimal decision variable $u^*(t, x)$, while the second, $\mathcal{Y}_\theta$, predicts the corresponding state variables $y(t, x)$ using a discrete-time neural operator architecture. Here, ω and θ denote the trainable parameters of $\mathcal{U}$ and $\mathcal{Y}$, respectively. The two components are optimized in two stages. First, $\mathcal{Y}_\theta$ is learned from numerical PDE trajectories. Its parameters are then fixed while the surrogate controller $\mathcal{U}_\omega$ is optimized end-to-end through differentiable rollouts of the learned dynamics using a primal-dual objective derived from Problem (1).

Learning Goal. PDE-OP trains on a distribution Π of problem instances generated by different states $y(t, x)$ for various t and x values. For instance, in the thermal regulation example, this distribution corresponds to the variety of temperature profiles that arise from different control inputs. With reference to (1a), the learning objective is:

$$\underset{\omega,\theta}{\text{Minimize}}\ \mathbb{E}_{y(t,x)\sim\Pi}\left[\mathcal{J}\left(\mathcal{U}_\omega(t, x, \hat{y}), \mathcal{Y}_\theta(t, x, \hat{u})\right)\right] \tag{2a}$$

$$\texttt{s.t.}\ \ \text{(1b)–(1f)}, \tag{2b}$$

where, to simplify the notation, we denote the estimated control variable as $\hat{u}(t, x) = \mathcal{U}_\omega(t, x, \hat{y})$, while the state variable estimate is obtained by recursive rollout and denoted as $\hat{y}(t, x) = \mathcal{Y}_\theta(t, x, \hat{u})$. The constraints are incorporated during controller training through a primal–dual formulation, which is described in Section 4.3. Given the complexity of solving Problem (2), the goal is to develop a fast and accurate neural PDE optimization surrogate.

4.1 Surrogate Controller

The first objective is to establish a neural network-based mapping that transforms the system state $y(t, x)$ of a PDE-constrained optimization problem (1) to a control action $u(t, x)$. Practically, this

mapping is implemented as a neural network $\mathcal{U}_\omega$, which takes the current estimate of the system state $\hat{y}(t,x)$ and desired terminal state $y_{\text{target}}(x)$ as input, and outputs an array of coefficients $\hat{\boldsymbol{c}}(t)$ that are then used to estimate the corresponding optimal decision variables $u^*(t,x)$. These coefficients weight a set of basis functions used to represent the control action, as described next.

Basis function representation of the control action. To parameterize the control action, we represent the spatial control profile $u(t,x)$ as a linear combination of continuous basis functions, with the corresponding coefficients predicted by a neural network. Similar parameterizations have been considered in Li et al. (2024); Feng et al. (2025). At each collocation time t_k, we write

$$u(t_k,x) = \sum_{i=1}^{M} c_i(t_k)\phi_i(x), \qquad t_k = 0,\dots,N_t, \tag{3}$$

where $\boldsymbol{c}(t_k) = [c_1(t_k),\dots,c_M(t_k)]^\top$ denotes the coefficients predicted by the controller $\mathcal{U}_\omega$.

This representation provides a structured, finite-dimensional parameterization of the spatial control profile. Rather than predicting the control independently at every spatial location, the controller predicts the coefficients $\boldsymbol{c}(t_k)$, while the continuous basis functions reconstruct $u(t_k,x)$ over the spatial domain. The basis expansion therefore acts as an inductive bias on the admissible control functions and can substantially reduce the dimension of the controller output when M is smaller than the number of spatial discretization points.

In our experiments, we use trigonometric basis functions e.g., $\phi_1(x) = \sin(\pi x/X)$, $\phi_2(x) = \sin(2\pi x/X)$, …, $\phi_M(x) = \sin(M\pi x/X)$. This Fourier-like representation is particularly suitable when the optimal control is spatially smooth, as encouraged by the regularization term in the objective. In such cases, a relatively small number of low-frequency modes can capture the dominant structure of the control signal. Increasing M enlarges the representational capacity of the controller and therefore allows more complex spatial control profiles to be approximated.

Importantly, the basis representation is a modeling choice rather than a requirement of PDE-OP. Appendix 7.11.2 shows that directly predicting the control without a basis expansion yields qualitatively similar solution quality, indicating that the main computational gains arise from the proposed learning-based optimization framework rather than from this particular parameterization. The basis representation instead provides a compact and structured control parameterization. As shown in Appendix 7.11.3, increasing M improves approximation accuracy while having little effect on inference time due to parallel GPU computation.

Surrogate controller model training. The role of the controller surrogate $\mathcal{U}_\omega$ is to predict the array of coefficients $\boldsymbol{c}(t_k) = [c_1(t_k)\ \dots\ c_M(t_k)]^T$. Formally, the network $\mathcal{U}_\omega$ approximates the function

$$U : \big(y(t_k,x), y_{\text{target}}(x)\big) \mapsto \boldsymbol{c}(t_k), \tag{4}$$

which maps the state variable $y(t_k,x)$ and desired terminal state $y_{\text{target}}(x)$ to the corresponding controller weights $\boldsymbol{c}(t_k)$, subsequently used to construct the control action $u(t_k,x)$ according to (3). The surrogate controller model $\mathcal{U}_\omega$ is trained *on-the-fly*, using a dataset $\mathcal{D} = \{(\hat{\boldsymbol{y}}^i(t_k,x),\ y_{\text{target}}(x))\}_{i=1}^N$, where $\hat{\boldsymbol{y}}(t_k,x)$ denotes the state variables estimated by the dynamic predictor $\mathcal{Y}_\theta$ and N is the number of samples. Here, bold symbols $\boldsymbol{y}(t_k,x)$ denote the space-discrete version of the corresponding state $y(t_k,x)$. Note that this setting is *self-supervised*: optimal weights $\boldsymbol{c}(t_k)$ are unknown, and candidate weights $\hat{\boldsymbol{c}}(t_k) = \mathcal{U}_\omega(\hat{\boldsymbol{y}}(t_k,x), y_{\text{target}}(x))$ are evaluated solely through the value of the objective $\mathcal{J}$ in (1a) that they induce. Figure 1 illustrates the interaction between PDE-OP's proxy optimizer $\mathcal{U}_\omega$ and dynamics predictor $\mathcal{Y}_\theta$ at time $t = t_k$. Given the current predicted state $\hat{y}(t_k,x)$, with $\hat{y}(t_0,x) = y_0(x)$, the surrogate controller predicts the coefficient vector $\boldsymbol{c}(t_k)$, which is used to construct the control action $u(t_k,x)$. The dynamics predictor then takes the current state and control as inputs and predicts the next state $\hat{y}(t_{k+1},x)$. Repeating this procedure over the time horizon produces the full predicted state and control trajectories.

To construct our surrogate controller, we build on the proxy optimizer method proposed by Park & Van Hentenryck (2023), which allows self-supervised training without access to precomputed optimal solutions, while directly encouraging feasibility through primal–dual updates. Once trained, the model $\mathcal{U}_\omega$ can be used to generate near-optimal solutions in real time. The next section discusses how the state variables are estimated using the proposed time-discrete Neural Operator architecture.

4.2 Discrete-time Neural Operator

To capture the PDEs governing the system dynamics, the paper proposes a discrete-time Neural Operator, here denoted as $\mathcal{Y}_\theta$, which models the PDE solution operator. The proposed dynamic predictor employs two subnetworks: the first, which we denote as *branch network*, encodes input functions (e.g., control signals), while the second, called *trunk network*, encodes the coordinates (e.g., spatial locations) where the solution is evaluated. Their outputs are combined to approximate the target operator. This architecture is inspired by (Lu et al., 2021), a neural operator framework designed to approximate mappings between infinite-dimensional function spaces and differs from recent time-integrated neural operator work such as TI-DeepONet (Nayak & Goswami, 2026) which learns a temporal-derivative field and advances the solution through a numerical time-integration scheme, which introduces computational overhead.

Recall that the PDE-constrained problem couples system dynamics and decision variables. To take account for this coupling, the proposed architecture incorporates both the system state and the controller output. Given the current estimated state $\hat{\boldsymbol{y}}(t_k, x)$ and the control weights $\hat{\boldsymbol{c}}(t_k)$ estimated by $\mathcal{U}_\omega$, the network $\mathcal{Y}_\theta$ outputs an estimate of the state at the subsequent collocation point in time:

$$\hat{\boldsymbol{y}}(t_{k+1}, x) = \mathcal{Y}_\theta(t, x, \hat{\boldsymbol{y}}(t_k, x), \hat{\boldsymbol{c}}(t_k)) \,. \tag{5}$$

Rolling out (5) across $k = 0, \ldots, T-1$ yields the full estimated state trajectory, $\{\hat{\boldsymbol{y}}(t_k, x)\}_{k=0}^{T}$.

Branch and trunk network. The branch net encodes the estimated system state sampled at n sensor locations, $\hat{\boldsymbol{y}}(t_k, x)$, together with estimated control weights $\hat{\boldsymbol{c}}(t_k)$, and maps this concatenated input to a d-dimensional vector of coefficients $\boldsymbol{b}(\hat{\boldsymbol{y}}(t_k, x), \hat{\boldsymbol{c}}(t_k)) \in \mathbb{R}^d$. The trunk net encodes the spatial coordinates x, producing a latent feature vector $\boldsymbol{\gamma}(x) \in \mathbb{R}^d$. The estimated subsequent state $\hat{\boldsymbol{y}}(t_{k+1}, x)$ is obtained by combining the branch coefficients and trunk features as

$$\hat{y}(t_{k+1}, x_i) = \sum_{j=1}^{d} b_j(\hat{\boldsymbol{y}}(t_k), \hat{\boldsymbol{c}}(t_k))\, \gamma_j(x_i) + \beta, \qquad i = 1, \ldots, n, \tag{6}$$

Model initialization and training. Since the proposed time-discrete neural operator model $\mathcal{Y}_\theta$ takes as input the basis-function weights estimated by the surrogate controller $\mathcal{U}_\omega$, it is practical to initialize the dynamic predictor accordingly. To this end, we initialize $\mathcal{Y}_\theta$ in a supervised fashion. The training data are generated by sampling time-varying control weight sequences $\boldsymbol{c}(t_k)$ from a Gaussian distribution, e.g., $c_i(t_k) \sim \mathcal{N}(0, \sigma_i)$, and specifying the initial condition $y(t_0, x)$, so as to expose $\mathcal{Y}_\theta$ to control inputs that are representative of those encountered during subsequent inference. The corresponding solution $y(t, x)$ satisfying (1b)–(1f) is computed using a numerical PDE solver. Each trajectory is discretized into one-step training samples of the form $(y(t_k, x), \boldsymbol{c}(t_k)) \mapsto y(t_{k+1}, x)$. The network $\mathcal{Y}_\theta$ is trained to minimize the prediction error between its outputs and the ground-truth:

$$\min_{\theta} \; \mathbb{E}_{(\boldsymbol{c}, y) \sim \mathcal{D}} \Big[\big\| \hat{y}(t, x) - y(t, x) \big\|^2 \Big], \tag{7}$$

where $\mathcal{D}$ denotes the dataset of pairs $(\boldsymbol{c}, y)$. This strategy ensures that $\mathcal{Y}_\theta$ learns to approximate the PDE solution operator given a distribution of weights $\boldsymbol{c}(t)$ that approximates those generated by the controller $\mathcal{U}_\omega$, enabling accurate PDE-solutions for the predicted weights $\hat{\boldsymbol{c}}(t)$ during inference.

4.3 Handling Static and Dynamics Constraints Jointly

To integrate the proposed time-discrete neural operator within the decision process, this paper proposes a primal–dual learning approach, which is inspired by the augmented Lagrangian relaxation technique (Hestenes, 1969) adopted in classical constrained optimization. In Lagrangian relaxation, some or all the problem constraints are relaxed into the objective function using Lagrangian multipliers to capture the penalties induced by violating them. The proposed formulation leverages Lagrangian duality to integrate trainable and weighted regularization terms that encapsulates violations of the constraint functions. When all the constraints are relaxed, the violation-based Lagrangian relaxation of problem (1) is

$$\begin{aligned} \min_{u} \quad & \mathcal{J}(u(t,x), y(t,x)) \\ & + \boldsymbol{\lambda}_{h'}^{\top} \big| \boldsymbol{h}'(u(t,x), y(t,x)) \big| \\ & + \boldsymbol{\lambda}_{g}^{\top} \max\big(\mathbf{0}, \boldsymbol{g}(u(t,x), y(t,x))\big), \end{aligned} \tag{8}$$

where $\mathcal{J}, \boldsymbol{g}$ are defined in (1a) and (1f), and $\boldsymbol{h}'$ is

$$\boldsymbol{h}'(u, y) = \begin{bmatrix} \partial_t y(t,x) - \mathcal{I}_t(y,u,t,x) \\ \partial_x y(t,x) - \mathcal{I}_x(y,u,t,x) \\ y(0,x) - y_0(x) \\ y(t,x) - y_\partial(t,x) \\ \boldsymbol{h}(u,y) \end{bmatrix} = \mathbf{0}. \tag{9}$$

It denotes the set of *all* equality constraints of problem (1), thus extending the constraints $\boldsymbol{h}$ in (1f), with the system dynamics (1b)-(1c) and the initial and boundary conditions equations (1d)-(1e) written in an implicit form as above. Therein, $\boldsymbol{\lambda}_{h'}$ and $\boldsymbol{\lambda}_g$ are the vectors of Lagrange multipliers associated with functions $\boldsymbol{h}'$ and $\boldsymbol{g}$. The key advantage of expressing the system dynamics (1b)-(1c) and initial and boundary conditions (1d)-(1e) in the same implicit form as the equality constraints $\boldsymbol{h}$ is treating the system dynamics in the same manner as the constraint functions $\boldsymbol{h}$ during training. This strategy does not guarantee exact feasibility at inference time, as such feasibility is evaluated explicitly in Section 5.4 and Appendix 8.

The proposed primal-dual learning method employs an iterative procedure to jointly optimize the surrogate controller parameters ω and the dual variables $\boldsymbol{\lambda}_{h'}$ and $\boldsymbol{\lambda}_g$. Specifically, it minimizes the following modified Lagrangian objective:

$$\begin{aligned} \mathcal{L}^{\text{PDE-OP}}(\hat{u}, \hat{y}) = \mathcal{J}(\hat{u}(t,x), \hat{y}(t,x)) \\ + \boldsymbol{\lambda}_{h'}^\top \left| \boldsymbol{h}'(\hat{u}(t,x), \hat{y}(t,x)) \right| \\ + \boldsymbol{\lambda}_g^\top \max\left(\mathbf{0}, \boldsymbol{g}(\hat{u}(t,x), \hat{y}(t,x))\right), \end{aligned} \tag{10}$$

where $\mathcal{J}(\hat{u}, \hat{y})$ denotes the objective function, while the remaining terms penalize equality and inequality constraint violations of the predicted control and state trajectories. At iteration $j+1$, the controller surrogate parameters are updated using stochastic gradient descent:

$$\omega^{j+1} = \omega^j - \eta \nabla_\omega \mathcal{L}^{\text{PDE-OP}}\left(\mathcal{U}_{\omega^j}^{\boldsymbol{\lambda}^j}, \mathcal{Y}\right), \tag{11}$$

where η is the learning rate. Rather than retraining the surrogate from scratch at each iteration, the parameters are initialized from the previous iterate, enabling the model to progressively adapt as the dual variables evolve.

Finally, the Lagrange multipliers are updated according to

$$\begin{aligned} \boldsymbol{\lambda}_{h'}^{j+1} &= \boldsymbol{\lambda}_{h'}^j + \rho \left| \boldsymbol{h}'(\hat{u}(t,x), \hat{y}(t,x)) \right|, \\ \boldsymbol{\lambda}_g^{j+1} &= \boldsymbol{\lambda}_g^j + \rho \max\left(\mathbf{0}, \boldsymbol{g}(\hat{u}(t,x), \hat{y}(t,x))\right), \end{aligned} \tag{12}$$

where ρ is the dual step size and the $\max$ operator is applied element-wise.

5 Experimental Setting

In this section, we evaluate the proposed method on three optimal control tasks of varying complexity. These problems are widely used for evaluating numerical and learning-based methods for PDE-constrained optimization tasks Bhan et al. (2024); Mowlavi & Nabi (2022); Sarkar et al. (2025); Tröltzsch (2010); Hinze et al. (2008). The benchmarks include voltage dynamics (1D linear PDE), the 2D heat equation (2D linear PDE), and the 1D Burgers' equation (1D nonlinear PDE), covering both time-invariant and time-varying control settings. All tasks impose box constraints on the control action, or on the basis weights where a basis representation is used. For space limitation, we defer the following additional experimental results to Appendix 7.10.1, which includes the predictive performance of PDE-OP's $\mathcal{Y}_\theta$ on a synthetic 1D Heat equation, a battery thermal management formulation with real-world parameters from Liu et al. (2019), experiments under complex target profiles, and a robustness study of the 1D Burgers' equation with spatially varying coefficients to evaluate robustness under parametric uncertainty.

PDE-OP is compared to state-of-the-art approaches for optimal control problems and learning-based method for operator learning: the Direct method (Betts, 2010), the Adjoint-Sensitivity method (Cao et al., 2003), linear and nonlinear Model Predictive Control (LMPC-NMPC) (Schwenzer et al.,

2021), Surrogate-based Trajectory Optimization (SBTO) (Hwang et al., 2022), and DeepONet MPC (de Jong et al., 2025). A comprehensive description of these methods is provided in Appendix 7.2–7.7. Unlike SBTO and DeepONet MPC, which learn a surrogate of the PDE dynamics but still rely on iterative optimization at inference time to compute the control action, PDE-OP directly predicts the control through a learned surrogate controller, without solving an optimization problem at inference time, requiring only an amortized neural rollout. Each baseline method is tuned separately to achieve its best performance, with implementation details reported in Appendix 10. For each experiment and method, we report the terminal-state MSE, the rollout MSE of $\mathcal{Y}_\theta$, inference/runtime, training time, and constraint violations. Rollout prediction errors, training times, and constraint-satisfaction results are summarized in Tables 3, 4, and 2, respectively, while Table 1 reports terminal-state MSE and inference time.

Table 1: Comparison across different tasks and methods. All results use sinusoidal target profiles. Further details and results for other complex target profiles can be found in Appendix 7.10.

PDE System	Control	Methods	Runtime	Accuracy
Voltage (React.-Diff.)	Open-Loop	Direct (FD)	39.284s	0.00710
		Adjoint	2.564s	0.00721
		SBTO	0.599s	0.00939
		PDE-OP	**0.035s**	**0.00695**
2D Heat (Linear)	Closed-Loop	LMPC	67.258s	**0.00712**
		Adjoint	6.349s	**0.00712**
		SBTO	15.483s	0.00724
		DeepONet MPC	17.888s	0.01868
		PDE-OP	**0.031s**	0.00721
1D Burgers' (Nonlinear)	Closed-Loop	NMPC	878.667s	0.00007
		Adjoint	103.014s	0.00023
		SBTO	67.391s	**0.00006**
		DeepONet MPC	98.371s	0.01392
		PDE-OP	**0.062s**	0.00030

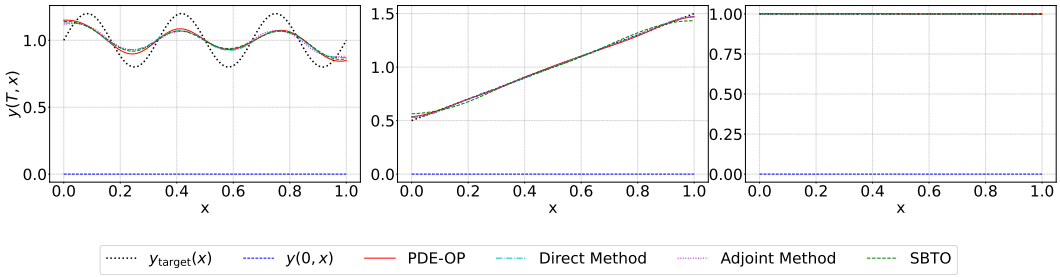

Figure 2: Voltage Equation. Comparison between PDE-OP and the baseline methods across three different target profiles: $y_{\text{target}}(x) = 1 + 0.2\sin(6\pi x)$ (left), $y_{\text{target}}(x) = x + 0.5$ (center), and $y_{\text{target}}(x) = 1$ (right).

5.1 Optimal Control of the Voltage Equation

Here we consider the problem of regulating voltage dynamics in a *leaky* electrical transmission line. The system is modeled by a controlled reaction–diffusion PDE with Neumann boundary conditions; the goal is to steer the voltage profile $y(t, x)$ toward a prescribed target distribution $y_{\text{target}}(x)$ using a space-dependent control action $u(x)$. We refer to Appendix 7.9.1 for a mathematical description of the problem, datasets, and related specifics.

Results. As shown in Table 1 and Figure 2, PDE-OP is simultaneously the fastest and the most accurate method on this task: it runs in 0.035s ($\sim 1.1 \times 10^3\times$ faster than Direct; $\sim 73\times$ faster than Adjoint; $\sim 17\times$ faster than SBTO), with an MSE of 6.95×10^{-3}. The Direct method is the slowest, requiring 39.284s for a terminal MSE of 7.10×10^{-3}. The Adjoint method is considerably faster at 2.564s and slightly less accurate (MSE $= 7.21 \times 10^{-3}$), while SBTO is faster still at 0.599s but yields the largest error of the four (MSE $= 9.39 \times 10^{-3}$). These results show that the proposed approach incurs small approximation errors from the use of a surrogate predictor and a

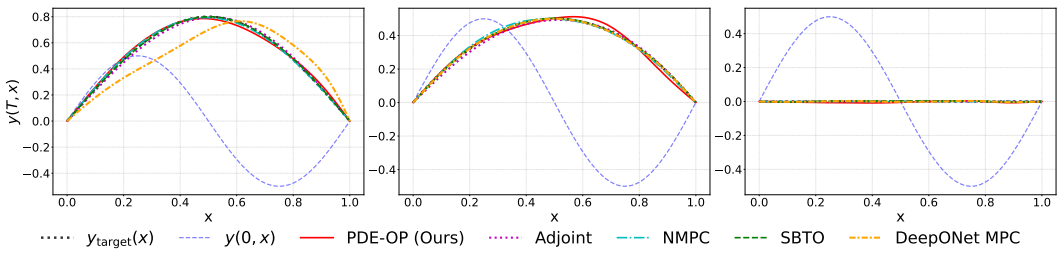

Figure 3: 1D Burgers' Equation. Comparison between PDE-OP and the baseline methods across three different target profiles: $y_{\text{target}}(x) = 0.8(\sin(\pi x))$ (left), $y_{\text{target}}(x) = 2x(1-x)$ (center), and $y_{\text{target}}(x) = 0$ (right).

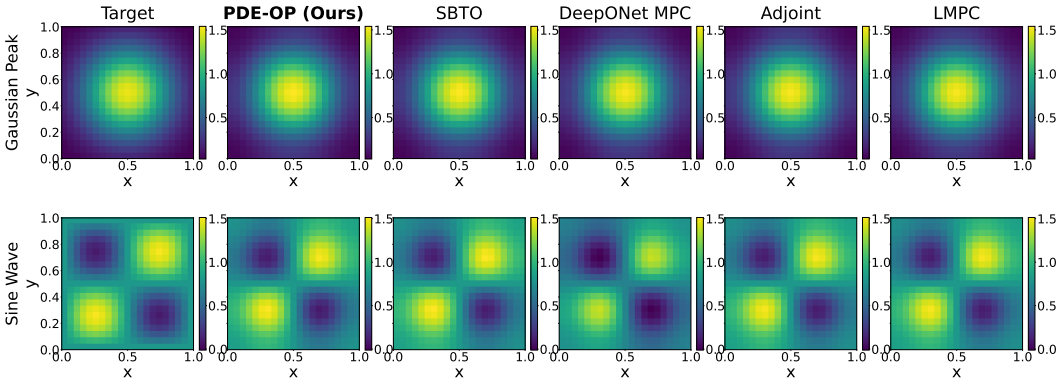

Figure 4: 2D Heat Equation. Comparison between PDE-OP and the baseline methods across two different target profiles: *Gaussian Peak(upper)*, *Sine Wave(lower)*.

proxy optimizer, while producing solutions that are comparable to the numerical baselines and one to four orders of magnitude faster.

5.2 Optimal Control of the 2D Heat Equation

Next, we consider the task of controlling the temperature distribution in a 2D domain, modeled by a reaction–diffusion equation. Here, $u(t, \boldsymbol{x})$ is a spatiotemporal function. Starting from an initial temperature profile, the objective is to drive the system toward a prescribed target distribution while minimizing control effort. We refer to Appendix 7.9.3 for a mathematical description of this problem, datasets, and related specifics.

Results. As shown in Table 1 and Figure 4, given a sinusoidal target, PDE-OP provides a favorable trade-off, with an MSE of 7.21×10^{-3} and an inference time of 0.031s, yielding speedups ranging from $200\times$ to nearly $2200\times$. The Adjoint and LMPC generate state trajectories that result in a terminal MSE of 7.12×10^{-3} each, running in 6.35s and 67.26s, respectively. DeepONet MPC produces an MSE of 1.87×10^{-2} and runs in 17.9s, while SBTO reaches a terminal MSE of 7.24×10^{-3}, and runs in 15.5s.

5.3 Optimal Control of the 1D Burgers' Equation

Here we consider the problem of controlling the 1D viscous Burgers' equation with Dirichlet boundary condition. This equation is a standard benchmark in physics-informed machine learning as it models fluid dynamics phenomena like shock waves and turbulence, arising from the interplay between nonlinear advection and linear diffusion. We refer to Appendix 7.9.4 for a mathematical description of the problem, datasets, and related specifics. As in the previous task, the control action is a function of time and space.

Table 2: Constraint satisfaction of PDE-OP over 40 unseen instances.

PDE System	Replay MSE	PDE Resid.	BC Err.	Mean Control Violation	Violation Rate (%)
Voltage	1.09×10^{-3}	7.08×10^{-2}	2.92×10^{-1}	2.26×10^{-2}	0.875%
2D Heat	4.92×10^{-2}	7.36×10^{-3}	5.42×10^{-2}	2.20×10^{-1}	1.36%
1D Burgers	6.85×10^{-3}	8.19×10^{-3}	7.70×10^{-5}	8.07×10^{-3}	0.731%

Results. As shown in Table 1 and Figure 3, PDE-OP produces qualitatively comparable predictions with the best methods (MSE $= 3.0 \times 10^{-4}$), while running in only 0.062s approximately $1.42 \times 10^4 \times$ faster than NMPC and $1.66 \times 10^3 \times$ faster than the Adjoint method. In contrast, the learning-based baseline SBTO achieves the lowest error (MSE $= 5.7 \times 10^{-5}$) followed by NMPC (MSE $= 7.0 \times 10^{-5}$), but they run in 67.391s and 878.667s, respectively. DeepONet MPC struggles with accuracy (MSE $= 1.39 \times 10^{-2}$), while the Adjoint method yields an MSE of 2.3×10^{-4}, running in 103.014s.

Table 3: Prediction error of PDE-OP's dynamic predictor Y_θ.

PDE System	MSE
Voltage Control	3.72×10^{-6}
2D Heat Equation	1.04×10^{-4}
1D Burgers' Equation	1.05×10^{-5}

Table 4: Training time (in seconds) of PDE-OP components on one GPU.

PDE System	$\mathbf{Y}_\theta$	$\mathbf{U}_\omega$
Voltage Control	1356	6474
2D Heat Equation	12018	3546
1D Burgers' Equation	12660	15240

5.4 Constraint Satisfaction and Physical Consistency

To evaluate the feasibility of the predicted control actions, we plug the controls generated by PDE-OP in the reference numerical PDE solver and measure the resulting terminal error and constraint violations on 40 unseen instances per benchmark. Specifically, we report the terminal error

$$e_{\text{true}} = \frac{1}{N_x} \|y(T) - y_{\text{target}}\|_2^2,$$

where $y(T)$ denotes the terminal state obtained by simulating the predicted control with the numerical solver. We also measure the PDE residual, boundary-condition error, and mean control bound violations and violation rates. The details can be found in Appendix 8. Table 2 summarizes constraint satisfaction over 40 unseen instances. Voltage achieves the lowest replay MSE, while Burgers exhibits the smallest boundary-condition error and the lowest control-violation magnitude and rate.

These results demonstrate the potential of the proposed method, which, particularly in complex settings with nonlinear dynamics such as Burgers' equation achieves speedups of three to four orders of magnitude while maintaining solution quality comparable to state-of-the-art numerical methods. Together with the constraint-satisfaction and physical-consistency results above, these findings highlight the potential of PDE-OP for real-time deployment.

6 Conclusion

This work was motivated by the efficiency requirements associated with solving optimization problems constrained by partial differential equations. We introduced PDE-OP, a learning-based framework that integrates a surrogate controller with a dynamic predictor to approximate solutions to PDE-constrained optimization problems in real time. The proposed dual-network architecture uses a pretrained dynamic predictor that is kept fixed while the control policy is trained through differentiable rollouts; the primal–dual objective penalizes violations of the governing dynamics and problem constraints during controller training. Numerical results across multiple PDE-constrained optimal control tasks demonstrate that PDE-OP achieves solution quality comparable to state-of-the-art numerical methods while providing speedups of up to four orders of magnitude. These results highlight the potential of learning-based optimization surrogates for enabling real-time decision making in scientific and engineering applications governed by PDEs.

AI USE STATEMENT

We used generative AI tools to improve the clarity and presentation of the manuscript, refine the visual presentation of figures, and assist with parts of the literature review. All AI-assisted text, figures, and literature references were reviewed and verified by the authors. Generative AI was not used to generate experimental data or determine the scientific conclusions of this work. The authors take full responsibility for the paper's content, including all claims and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personal or sensitive data, or experiments affecting individuals or communities. The proposed methodology is evaluated on standard optimization benchmarks and is not intended for harmful or discriminatory applications. We are not aware of any specific concerns related to privacy, security, fairness, or conflicts of interest arising from this research.

REPRODUCIBILITY STATEMENT

The paper and appendix provide the problem formulations, model architecture, training procedure, evaluation metrics, theoretical assumptions. Additional implementation details, dataset-generation procedures, hyperparameters, and solver configurations are included in the appendix. An anonymized implementation will be provided with the supplementary materials, and the complete source code will be released publicly upon publication.

REFERENCES

John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2 edition, 2010.

Luke Bhan, Yuexin Bian, Miroslav Krstic, and Yuanyuan Shi. Pde control gym: A benchmark for data-driven boundary control of partial differential equations, 2024. URL https://arxiv.org/abs/2405.11401.

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

Yang Cao, Shengtai Li, Linda Petzold, and Radu Serban. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint dae system and its numerical solution. *SIAM Journal on Scientific Computing*, 24(3):1076–1089, 2003.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 6572–6583. Curran Associates Inc., 2018.

John Crank and Phyllis Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical proceedings of the Cambridge philosophical society*, volume 43, pp. 50–67. Cambridge University Press, 1947.

Thomas Oliver de Jong, Khemraj Shukla, and Mircea Lazar. Deep operator neural network model predictive control, 2025. URL https://arxiv.org/abs/2505.18008.

Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Markus A Dihlmann and Bernard Haasdonk. Certified pde-constrained parameter optimization using reduced basis surrogate models for evolution problems. *Computational Optimization and Applications*, 60:753–787, 2015.

Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

Mingquan Feng, Zhijie Chen, Yixin Huang, Yizhou Liu, and Junchi Yan. Optimal control operator perspective and a neural adaptive spectral method. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. ISBN 978-1-57735-897-8. doi: 10.1609/aaai.v39i14.33596. URL https://doi.org/10.1609/aaai.v39i14.33596.

Ferdinando Fioretto, Pascal Van Hentenryck, Terrence W.K. Mak, Cuong Tran, Federico Baldo, and M. Lombardi. Lagrangian duality for constrained deep learning. In *ECML/PKDD*, 2020. URL https://api.semanticscholar.org/CorpusID:214802938.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Magnus R Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4:303–320, 1969.

Michael Hinze, René Pinnau, Michael Ulbrich, and Stefan Ulbrich. *Optimization with PDE Constraints*, volume 23 of *Mathematical Modelling: Theory and Applications*. Springer, 2008. doi: 10.1007/978-1-4020-8839-1.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Rakhoon Hwang, Jae Yong Lee, Jin Young Shin, and Hyung Ju Hwang. Solving pde-constrained control problems using operator learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4504–4512, 2022.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4475–4482, 2021. doi: 10.24963/ijcai.2021/610. URL https://doi.org/10.24963/ijcai.2021/610.

James Kotary, Vincenzo Di Vito, Jacob Cristopher, Pascal Van Hentenryck, and Ferdinando Fioretto. Learning joint models of prediction and optimization, 2024. URL https://arxiv.org/abs/2409.04898.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces, 2024. URL https://arxiv.org/abs/2108.08481.

Dieter Kraft. Algorithm 733: Tomp–fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software (TOMS)*, 20:262 – 281, 1994.

Zhexian Li, Athanassios S Fokas, and Ketan Savla. On linear quadratic regulator for the heat equation with general boundary conditions. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 7594–7599. IEEE, 2024.

Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 6755–6766, 2020.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL https://arxiv.org/abs/2010.08895.

Yiqun Liu, Y. Gene Liao, and Ming-Chia Lai. Transient temperature distributions on lithium-ion polymer sli battery. *Vehicles*, 1(1):127–137, 2019. ISSN 2624-8921. doi: 10.3390/vehicles1010008. URL https://www.mdpi.com/2624-8921/1/1/8.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL http://dx.doi.org/10.1038/s42256-021-00302-5.

D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(99)00214-9. URL https://www.sciencedirect.com/science/article/pii/S0005109899002149.

Saviz Mowlavi and Saleh Nabi. Optimal control of pdes using physics-informed neural networks, 2022. URL https://arxiv.org/abs/2111.09880.

Dibyajyoti Nayak and Somdatta Goswami. Ti-deeponet: Learnable time integration for stable long-term extrapolation, 2026. URL https://arxiv.org/abs/2505.17341.

Roussel Desmond Nzoyem Ngueguin, David A.W. Barton, and Tom Deakin. A comparison of mesh-free differentiable programming and data-driven strategies for optimal control under pde constraints. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W 2023, pp. 21–28. ACM, November 2023. doi: 10.1145/3624062.3626078. URL http://dx.doi.org/10.1145/3624062.3626078.

Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 4052–4060, 2023.

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.

Dibakar Roy Sarkar, Ján Drgoňa, and Somdatta Goswami. Learning to control pdes with differentiable predictive control and time-integrated neural operators, 2025. URL https://arxiv.org/abs/2511.08992.

Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: an engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117:1327 – 1349, 2021.

Fredi Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*, volume 112. American Mathematical Soc., 2010.

Vincenzo Di Vito, Mostafa Mohammadian, Kyri Baker, and Ferdinando Fioretto. Learning to solve differential equation constrained optimization problems, 2024. URL https://arxiv.org/abs/2410.01786.

Matthew J Zahr and Charbel Farhat. Progressive construction of a parametric reduced-order model for pde-constrained optimization. *International Journal for Numerical Methods in Engineering*, 102(5):1111–1135, 2015.

7 APPENDIX

7.1 RELATED WORK

A central paradigm for decision-making coupled with dynamical systems is *Model Predictive Control (MPC)* Mayne et al. (2000). It repeatedly solves an optimization problem over a finite time horizon using predicted system states. While MPC provides high-quality control decisions, its reliance on repeatedly solving an optimization problem makes it computationally intractable and limits its applicability in real-time settings. In a similar fashion, exact PDE-constrained optimization methods rely on time–space discretization of the governing equations, after which the resulting high-dimensional system is incorporated into an optimization problem Hinze et al. (2008); Tröltzsch (2010). While theoretically well-founded, these approaches become quickly impractical as the problem size grows, especially in the presence of nonlinear dynamics or when real-time feasibility is required. To mitigate these limitations, surrogate models have been developed for reduced-order adjoint solutions Zahr & Farhat (2015), sensitivity PDEs Dihlmann & Haasdonk (2015), and learning PDE operators Hwang et al. (2022), offering significant computational advantages. Nevertheless, these methods often face trade-offs between accuracy and efficiency and limited adaptability across operating regimes Kovachki et al. (2024); Lu et al. (2021).

Differentiable programming frameworks have recently gained traction as an alternative route to PDE-constrained optimization, enabling end-to-end differentiation through numerical solvers using automatic differentiation (see Nzoyem Ngueguin et al. (2023) for a comparative study). These approaches typically follow a discretize-then-optimize paradigm and provide accurate gradients through the PDE solver itself. Recent work has also explored learning-based surrogates within differentiable predictive control frameworks for PDE systems, using time-integrated neural operators to learn neural control policies in an offline, self-supervised manner Sarkar et al. (2025). PDE-OP differs by directly learning a time-discrete state-transition operator, rather than a temporal-derivative model coupled with a numerical integrator, and by incorporating a primal–dual formulation for handling dynamics and operational constraint violations during controller training.

In parallel, to approximate solutions of constrained optimization problems in real-time setting, machine learning researchers have developed a class of methods known as *proxy optimizers*. These methods employ machine learning models, to learn mappings from problem parameters or system configurations to optimal decisions, thereby significantly accelerating inference Kotary et al. (2021) compared to classical optimization solvers. Both supervised approaches Fioretto et al. (2020), which rely on precomputed optimal solutions, and self-supervised ones Park & Van Hentenryck (2023), which exploit knowledge of the problem structure, have been proposed, showing promising results. A recurring challenge in this setting is ensuring feasibility: learned solutions may violate problem constraints. This issue has been tackled through penalty-based training Fioretto et al. (2020), implicit-layer formulations that embed constraints directly into the model architecture Donti et al. (2021), or post-processing projections to restore feasibility Kotary et al. (2024).

A large body of recent work has also focused on *learning surrogates for PDE dynamics*. Physics-Informed Neural Networks Raissi et al. (2019) directly encode PDE residuals into the loss function, enabling solutions that adhere to the governing equations. Neural operator approaches, such as the Fourier Neural Operator Li et al. (2021) and DeepONet Lu et al. (2021), learn mappings between infinite-dimensional function spaces, which allows them to approximate solution operators of *families of PDEs*, with strong generalization across different boundary conditions and coefficients Kovachki et al. (2024). Other operator-learning frameworks Li et al. (2020); Bhattacharya et al. (2021) and kernel-based methods have also been developed. While these approaches excel at reproducing PDE solutions, they are typically not designed to incorporate decision variables or to directly solve PDE-constrained optimization problems. The closest related work is Vito et al. (2024), where the authors propose a learning-based surrogate solver to tackle ODE or SDE-constrained optimization tasks. *Our approach goes further, by integrating operator-learning surrogates within the optimization pipeline, thereby coupling the representation of PDE dynamics with proxy optimizers and enabling real-time solutions to PDE-constrained optimization tasks.*

7.2 Direct Method (Finite-Difference Approach)

The direct method treats the simulator as a black box mapping the bounded control vector $\boldsymbol{u}$ to the discrete objective $J(\boldsymbol{u})$ and employs a standard optimizer to minimize it. Here the control variable is discretized at each collocation point: $\boldsymbol{u} \in \mathbb{R}^{N_x}$.

Mechanism. We use L-BFGS-B Byrd et al. (1995) with per–coordinate box bounds $u_{\min} \le u_j \le u_{\max}$ to minimize (27). No analytic gradients are provided; instead, numerical derivatives are queried via finite differences. With a forward-difference method,

$$\frac{\partial J}{\partial u_j} \;\approx\; \frac{J(\boldsymbol{u}+\epsilon_j \boldsymbol{e}_j) - J(\boldsymbol{u})}{\epsilon_j}, \qquad \epsilon_j = \eta\,\big(1+|u_j|\big), \tag{13}$$

where $\boldsymbol{e}_j$ is the j-th unit vector and η is a small scalar (e.g., $\mu = 10^{-6}$). For higher accuracy (to which corresponds a double computational cost), the central-difference method is adopted:

$$\frac{\partial J_h}{\partial u_j} \;\approx\; \frac{J_h(\boldsymbol{u}+\epsilon_j \boldsymbol{e}_j) - J_h(\boldsymbol{u}-\epsilon_j \boldsymbol{e}_j)}{2\epsilon_j}. \tag{14}$$

Computational cost. Each gradient evaluation requires N_x+1 simulator calls with forward differences (or $2N_x$ with central differences), where N_x denotes the number of spatial collocation points. Since one simulator call is a complete rollout (22) over N_t time steps, the complexity per iteration is $\mathcal{O}\big((N_x+1)\,N_t\,N_x\big)$. This method is robust and simple, but its cost scales linearly with N_x, which becomes prohibitive for dense spatial discretizations.

7.3 Adjoint Sensitivity Method for Open-Loop Problem

The Adjoint-Sensitivity Method efficiently computes exact gradients of PDE-constrained objectives w.r.t. high-dimensional control variables. Crucially, the cost of one gradient evaluation is *one forward solve + one backward adjoint solve*, regardless of the number of control parameters. We derive the continuous adjoint for the open-loop, time-invariant Voltage Control Optimization task of the main text and then we present the discrete-time adjoint state used in our Crank–Nicolson (CN) implementation.

Function spaces & BCs. Consider $u \in L^2(\Omega)$, $y \in L^2([0,T]\times\Omega)$, and impose homogeneous Neumann BCs on V so that the boundary terms vanish in the variational calculus; the adjoint inherits the same Neumann BCs.

The problem consists of finding $u(x)$ minimizing

$$J(u) = \tfrac{1}{2}\int_\Omega (y(T,x) - y_{\text{target}}(x))^2\,dx \;+\; \tfrac{\gamma}{2}\int_\Omega u(x)^2\,dx, \tag{15}$$

subject to

$$\frac{\partial y}{\partial t} - D\frac{\partial^2 y}{\partial x^2} + \beta\big(y - y_{\text{ref}}(x)\big) - \alpha u(x) = 0, \tag{16a}$$

$$y(0,x) = y_0(x), \tag{16b}$$

$$\partial_x y(t,x) = 0, \tag{16c}$$

for $(t,x) \in [0,T]\times\Omega$, $x \in \Omega$, and $(t,x) \in [0,T]\times\partial\Omega$, respectively.

where $\gamma > 0$ is the regularization weight in the control effort.

Lagrangian. We define

$$\begin{aligned}\mathcal{L}(u,y,p) = J(u) + \int_0^T\!\!\int_\Omega p(t,x)\Big[\frac{\partial y}{\partial t} - D\frac{\partial^2 y}{\partial x^2} \\ + \beta\big(y - y_{\text{ref}}(x)\big) - \alpha u(x)\Big]\,dx\,dt.\end{aligned} \tag{17}$$

where $p(t,x)$ is the adjoint state. By imposing $\nabla_y J = 0$ and integrating by parts in t and x (with Neumann BCs) yields the adjoint system.

Adjoint PDE.

$$-\frac{\partial p}{\partial t} - D\frac{\partial^2 p}{\partial x^2} + \beta\, p = 0, \qquad \partial_x p|_{\partial\Omega} = 0. \tag{18}$$

Terminal condition.

$$p(T,x) = y(T,x) - y_{\text{target}}(x). \tag{19}$$

Gradient w.r.t. $u(x)$. Because u is time-invariant, variation w.r.t. u gives

$$\nabla_{u(x)} J \;=\; \gamma\, u(x) \;-\; \alpha \int_0^T p(t,x)\, dt. \tag{20}$$

We optimize $\boldsymbol{u}$ with L-BFGS-B under elementwise bounds $u_{\min} \le u_i \le u_{\max}$; no projection step is needed.

Semi-discrete forward dynamics. To simplify the notation, let $\boldsymbol{y}(t) = \begin{bmatrix} y(t,x_1) & \dots & y(t,x_i) & \dots & y(t,x_{N_x}) \end{bmatrix} \in \mathbb{R}^{N_x}$ be the discretized version of $y(t,x)$ on a uniform grid with Neumann BCs, $\boldsymbol{L} \in \mathbb{R}^{N_x \times N_x}$ be the the Neumann Laplacian, and $\boldsymbol{u} \in \mathbb{R}^{N_x}$ the *static* control vector with elements $u(x_i)$. Define $\boldsymbol{A} := D\,\boldsymbol{L} - \beta\, \boldsymbol{I}_{N_x}$ and $\boldsymbol{s} := \beta\, \boldsymbol{y}_{\text{ref}}$. Then

$$\dot{\boldsymbol{y}}(t) \;=\; \boldsymbol{A}\,\boldsymbol{y}(t) \;+\; \alpha\,\boldsymbol{u} \;+\; \boldsymbol{s}. \tag{21}$$

Crank–Nicolson (CN) step. With $\Delta t = T/N_t$ and $t_k = k\Delta t$, CN gives

$$\begin{aligned} \big(\boldsymbol{I}_{N_x} - \tfrac{\Delta t}{2}\boldsymbol{A}\big)\boldsymbol{y}(t_{k+1}) = \big(\boldsymbol{I}_{N_x} + \tfrac{\Delta t}{2}\boldsymbol{A}\big)\boldsymbol{y}(t_k) \\ + \Delta t\,\alpha\,\boldsymbol{u} + \Delta t\,\boldsymbol{s}. \end{aligned} \tag{22}$$

Equivalently, the affine one-step map is

$$\boldsymbol{y}(t_{k+1}) = \boldsymbol{A}_{\text{CN}}\,\boldsymbol{y}(t_k) \;+\; \boldsymbol{B}_{\text{stat}}\,\boldsymbol{u} \;+\; \boldsymbol{d}, \tag{23}$$

$$\boldsymbol{A}_{\text{CN}} = \big(\boldsymbol{I}_{N_x} - \tfrac{\Delta t}{2}\boldsymbol{A}\big)^{-1}\big(\boldsymbol{I}_{N_x} + \tfrac{\Delta t}{2}\boldsymbol{A}\big), \tag{24}$$

$$\boldsymbol{B}_{\text{stat}} = \Delta t\,\big(\boldsymbol{I}_{N_x} - \tfrac{\Delta t}{2}\boldsymbol{A}\big)^{-1}\,\alpha\,\boldsymbol{I}_{N_x}, \tag{25}$$

$$\boldsymbol{d} = \Delta t\,\big(\boldsymbol{I}_{N_x} - \tfrac{\Delta t}{2}\boldsymbol{A}\big)^{-1}\,\boldsymbol{s}. \tag{26}$$

Discrete objective. With $\boldsymbol{M} = \Delta x\, \boldsymbol{I}_{N_x}$ we have:

$$\begin{aligned} J_h(\boldsymbol{u}) &= \tfrac{1}{2}\,\|\boldsymbol{y}_{N_t} - \boldsymbol{y}_{\text{target}}\|_{\boldsymbol{M}}^2 + \tfrac{\gamma}{2}\,\|\boldsymbol{u}\|_{\boldsymbol{M}}^2 \\ &= \tfrac{1}{2}\,(\boldsymbol{y}_{N_t} - \boldsymbol{y}_{\text{target}})^\top \boldsymbol{M}(\boldsymbol{y}_{N_t} - \boldsymbol{y}_{\text{target}}) + \tfrac{\gamma}{2}\,\boldsymbol{u}^\top \boldsymbol{M}\boldsymbol{u}. \end{aligned} \tag{27}$$

Discrete adjoint & gradient w.r.t. static $\boldsymbol{u}$.

$$\begin{aligned} \boldsymbol{\lambda}_{N_t} &= \boldsymbol{M}\big(\boldsymbol{y}_{N_t} - \boldsymbol{y}_{\text{target}}\big), \\ \boldsymbol{\lambda}_k &= \boldsymbol{A}_{\text{CN}}^\top\,\boldsymbol{\lambda}_{k+1}, \qquad k = N_t - 1, \dots, 0. \end{aligned} \tag{28}$$

$$\begin{aligned} \nabla_{\boldsymbol{u}} J_h &= \gamma\,\boldsymbol{M}\,\boldsymbol{u} - \sum_{k=0}^{N_t-1} \boldsymbol{B}_{\text{stat}}^\top\,\boldsymbol{\lambda}_{k+1} \\ &= \gamma\,\boldsymbol{M}\,\boldsymbol{u} - \alpha\,\Delta t\,\big(\boldsymbol{I}_{N_x} - \tfrac{\Delta t}{2}\boldsymbol{A}\big)^{-\top}\left(\sum_{k=0}^{N_t-1}\boldsymbol{\lambda}_{k+1}\right). \end{aligned} \tag{29}$$

Computational cost. One forward CN rollout (22) and one backward adjoint sweep (28) yield the exact discrete gradient (29); then, $\boldsymbol{u}$ is updated with L-BFGS-B under box bounds. In 1D, the CN matrix is time-invariant and can be prefactored once, so that each step reduces to cheap triangular-matrix solver call; the total cost per iteration is $\mathcal{O}(N_t N_x)$.

7.4 TIME-VARYING ADJOINT FOR CLOSED-LOOP BASELINES

The adjoint method extends naturally to a time-varying control $u(t, x)$. The continuous adjoint PDE and terminal/boundary conditions remain as per Eq. (18)–(19), while the gradients w.r.t. $u(t, x)$ are calculated as:

$$\nabla_{u(t,x)} J \;=\; \gamma\, u(t,x) \;-\; \alpha\, p(t,x). \tag{30}$$

We use this time-varying adjoint for the Heat and Burgers' experiments (please refer to the *Optimal Control of the 2D Heat Equation* and *Optimal Control of the 1D Burgers' Equation* sections of the main text); the discrete forms below match our CN implementations.

Heat optimization problem — CN discretization. With $\boldsymbol{y}(t_{k+1}) = \boldsymbol{A}\boldsymbol{y}(t_k) + \boldsymbol{B}\boldsymbol{u}(t_k) + \boldsymbol{g}$ and a quadratic running/terminal costs, the discrete adjoint equation and per-step gradient are

$$\boldsymbol{\lambda}_{N_t} = \boldsymbol{Q}\,(\boldsymbol{y}_{N_t} - \boldsymbol{y}_{\text{target}}), \tag{31}$$

$$\boldsymbol{\lambda}_k = \boldsymbol{A}^\top \boldsymbol{\lambda}_{k+1} + \boldsymbol{T}_Q\,(\boldsymbol{y}(t_k) - \boldsymbol{y}_{\text{target}}), \tag{32}$$

$$\nabla_{\boldsymbol{u}(t_k)} J_h = \boldsymbol{R}\,\boldsymbol{u}(t_k) + \boldsymbol{B}^\top \boldsymbol{\lambda}_{k+1}. \tag{33}$$

for $k = N_t - 1, \ldots, 0$.

This is the gradient used by the time-varying Adjoint Method in the 2D heat equation experiments. $\boldsymbol{Q}, \boldsymbol{T}_Q,$ and $\boldsymbol{R}$ represent the weight matrices.

Burgers' optimization problem — implicit CN step. We write one CN step as the implicit map $\boldsymbol{y}(t_{k+1}) = \mathcal{G}(\boldsymbol{y}(t_k), \boldsymbol{u}(t_k); \Delta t)$ with Dirichlet endpoints enforced at each step. Let

$$A_k \;=\; I - \tfrac{\Delta t}{2}\, J(\boldsymbol{y}(t_{k+1})), \qquad B_k \;=\; -I - \tfrac{\Delta t}{2}\, J(\boldsymbol{y}(t_k)), \tag{34}$$

where $J(\boldsymbol{y}) = -\boldsymbol{D}\,\mathrm{diag}(\boldsymbol{y}) + \nu\,\boldsymbol{D}^2$ is the Jacobian of the semi-discrete residual, and $\boldsymbol{B}$ stacks the actuator shapes. For running cost $L_k = \frac{\Delta t}{2}\|\boldsymbol{y}(t_{k+1}) - \boldsymbol{y}_{\text{target}}\|_2^2 + \frac{\alpha \Delta t}{2}\|\boldsymbol{u}(t_k)\|_2^2$, the closed-loop adjoint/backward sweep is

$$A_k^\top \boldsymbol{q}_k = \Delta t\,(\boldsymbol{y}(t_{k+1}) - \boldsymbol{y}_{\text{target}}), \tag{35}$$

$$\boldsymbol{p}_k = \Delta t\,(\boldsymbol{y}(t_k) - \boldsymbol{y}_{\text{target}}) - B_k^\top \boldsymbol{q}_k, \tag{36}$$

$$\nabla_{\boldsymbol{u}(t_k)} J_h = \Delta t\big(\alpha\,\boldsymbol{u}(t_k) + \boldsymbol{B}^\top \boldsymbol{q}_k\big), \tag{37}$$

which are the adjoint equations used in the 1D Burgers' equation experiments. Here q_k and p_k are the adjoint variables.

7.5 MODEL PREDICTIVE CONTROL (MPC)

In what follows, to simplify notation we omit the dependency from the spatial variable x. At each time instant t_k, MPC solves a finite-horizon problem of N_p steps given inputs $\{\boldsymbol{u}(t_i)\}_{i=k}^{k+N_p-1}$; it applies the first input $\boldsymbol{u}(t_k)$, then applies shifts and iterates(receding horizon).

Linear MPC for the Heat equation. With Crank–Nicolson discretization and cosine-basis expansion for the control action, the dynamics are linear:

$$\boldsymbol{y}(t_{i+1}) = \boldsymbol{A}\,\boldsymbol{y}(t_i) + \boldsymbol{B}\,\boldsymbol{c}(t_i) + \boldsymbol{g}. \tag{38}$$

Here, $\boldsymbol{y}(t_i) \in \mathbb{R}^{N_x}$ and $\boldsymbol{c}(t_i) \in \mathbb{R}^M$. We solve the convex QP

$$\begin{aligned} \min_{\{\boldsymbol{c}(t_i)\}_{i=k}^{k+N_p-1}} \quad & \sum_{i=k}^{k+N_p-1} \Big(\|\boldsymbol{y}(t_i) - \boldsymbol{y}_{\text{target}}\|_{\boldsymbol{T}_Q}^2 + \|\boldsymbol{c}(t_i)\|_{\boldsymbol{R}}^2\Big) \\ & + \|\boldsymbol{y}(t_{k+N_p}) - \boldsymbol{y}_{\text{target}}\|_{\boldsymbol{Q}}^2 \\ \text{s.t.} \quad & (38), \\ & c_{\min} \le \boldsymbol{c}(t_i) \le c_{\max}, \end{aligned} \tag{39}$$

where $i = k, \ldots, k+N_p-1$. Here, $\boldsymbol{A} = M_L^{-1} M_R$, $\boldsymbol{B} = M_L^{-1}(\Delta t\, \boldsymbol{B}_c)$, $\mathbf{g} = M_L^{-1}(\Delta t\, \mathbf{s})$ (Neumann BCs), $\mathbf{B}_c$ stacks $\cos(j\pi x/X)$; (39) is solved with CVXPY/OSQP Diamond & Boyd (2016). Here M_L and M_R are the CN matrices of the system, obtained from the semi-discrete PDE operator A_c:

$$M_L \;=\; I - \tfrac{\Delta t}{2}\, A_c, \qquad M_R \;=\; I + \tfrac{\Delta t}{2}\, A_c. \tag{40}$$

This way one CN step writes as:

$$M_L\, \boldsymbol{y}(t_{k+1}) \;=\; M_R\, \boldsymbol{y}(t_k) \;+\; \Delta t\, \boldsymbol{B}_c\, \boldsymbol{c}(t_k) \;+\; \Delta t\, \boldsymbol{s}. \tag{41}$$

Nonlinear MPC for Burgers' Equation. For viscous Burgers', the CN step is nonlinear due to convection term; we write one step as the implicit map

$$\boldsymbol{y}(t_{i+1}) \;=\; \mathcal{G}(\boldsymbol{y}(t_i), \boldsymbol{c}(t_i); \Delta t), \tag{42}$$

where $\mathcal{G}$ is the unique CN solution for Burgers' with input $\boldsymbol{B}\boldsymbol{c}(t_i)$ (computed via `fsolve` on the CN residual and by imposing Dirichlet endpoints at each step).

$$\begin{aligned} \min_{\{\boldsymbol{c}(t_i)\}_{i=k}^{k+N_p-1}} \quad & \sum_{i=k}^{k+N_p-1} \Big(\|\boldsymbol{y}(t_i) - \boldsymbol{y}_{\text{target}}\|^2_{\boldsymbol{T}_Q} + \|\boldsymbol{c}(t_i)\|^2_{\boldsymbol{R}} \Big) \\ & + \|\boldsymbol{y}(t_{k+N_p}) - \boldsymbol{y}_{\text{target}}\|^2_{\boldsymbol{Q}} \\ \text{s.t.} \quad & (42), \\ & c_{\min} \le \boldsymbol{c}(t_i) \le c_{\max}, \end{aligned} \tag{43}$$

, where $i = k, \ldots, k+N_p-1$. We use single shooting over the full-order model: propagate with (42), optimize with SLSQP Kraft (1994) under box bounds, apply $\mathbf{c}(t_k)$, advance one CN step, and iterate. Here $\mathbf{B}$ stacks $\sin((j+1)\pi x/X)$ shapes used in the implementation. Notably, $\mathbf{T}_Q, \mathbf{R}$, and $\mathbf{Q}$ represent the weight matrices, they are distinct from PDE-OP's learned constraint multipliers $\boldsymbol{\lambda}$.

7.6 Surrogate-Based Trajectory Optimization (SBTO)

We compare against a learning-based trajectory optimization baseline adapted from Hwang et al. (2022), which we refer to as Surrogate-Based Trajectory Optimization (SBTO). While Hwang et al. (2022) introduced a reconstruction regularizer to prevent adversarial artifacts in high-dimensional control spaces, we omit this term in our implementation. This is because we parameterize the control inputs using low-dimensional smooth basis functions (as described in the *Surrogate Controller* section of the main text), which inherently restricts the search space and acts as an implicit regularizer against high-frequency noise. Given an initial state $y(0, x)$ and a target state $y_{\text{target}}(x)$, the goal is to find the optimal sequence of time-varying weights $\boldsymbol{c}(t)$ that minimizes a cost function over the entire time horizon. The optimization problem can be defined as:

$$\begin{aligned} \min_{\{\boldsymbol{c}(t_i)\}_{i=0}^{N_t-1}} \quad & \|\boldsymbol{y}(t_{N_t}) - \boldsymbol{y}_{\text{target}}\|^2_{\mathbf{Q}} \\ & + \sum_{k=0}^{N_t-1} \Big(\|\boldsymbol{y}(t_{k+1}) - \boldsymbol{y}_{\text{target}}\|^2_{\mathbf{T_Q}} \\ & \qquad + \|\boldsymbol{c}(t_k)\|^2_{\mathbf{R}} \Big) \\ \text{s.t.} \quad & \boldsymbol{y}(t_{k+1}) = \mathcal{Y}_\theta(\boldsymbol{y}(t_k), \boldsymbol{c}(t_k)), \\ & \quad \text{for } k = 0, \ldots, N_t - 1 \\ & c_{\min} \le \boldsymbol{c}(t) \le c_{\max}, \end{aligned} \tag{44}$$

where $\mathbf{T}_Q, \mathbf{R}$, and $\mathbf{Q}$ represent the weight matrices This method consists of two distinct phases. **In Phase 1** a neural operator, here denoted as $\mathcal{Y}_\theta$, is trained offline to learn a differentiable surrogate for the PDE dynamics, mapping the current state and control to the next state. **In Phase 2** the surrogate model's parameters θ are frozen. For each new problem instance at test-time, an open-loop trajectory optimization is performed to find the optimal time-varying weights $\boldsymbol{c}(t)$ as described in (44). This problem is solved using a gradient-based optimizer (i.e., Adam) by backpropagating through the full unrolled trajectory estimated by the surrogate $\mathcal{Y}_\theta$.

7.7 DEEPONET MPC

DeepONet MPC is a closed-loop, receding-horizon strategy that combines operator learning with MPC. We follow the **Standard DeepONet MPC** formulation established as a benchmark in de Jong et al. (2025). Like SBTO, this method operates in two phases. **In Phase 1** the same differentiable surrogate $\mathcal{Y}_\theta$ is trained to learn the system dynamics. **In Phase 2**, however, instead of solving one large optimization problem as in SBTO, DeepONet MPC solves a new, smaller optimization problem at *each* time step t_k. Given the current measured state $\boldsymbol{y}(t_k)$, the controller plans a sequence of future actions $\boldsymbol{c}(t_k), \ldots, \boldsymbol{c}(t_{k+N_p-1})$ over a shorter planning horizon of length N_p. The optimization problem solved at step t_k is:

$$\begin{aligned}\min_{\{\boldsymbol{c}(t_{k+i})\}_{i=0}^{N_p-1}} \quad & \|\boldsymbol{y}(t_{k+N_p}) - \boldsymbol{y}_{\text{target}}\|^2_{\boldsymbol{T_Q}} \\ & + \sum_{i=0}^{N_p-1} \Big(\|\boldsymbol{y}(t_{k+i+1}) - \boldsymbol{y}_{\text{target}}\|^2_{\boldsymbol{Q}} \\ & \qquad + \|\boldsymbol{c}(t_{k+i})\|^2_{\boldsymbol{R}} \Big) \\ \text{s.t.} \quad & \boldsymbol{y}(t_{k+i+1}) = \mathcal{Y}_\theta(\boldsymbol{y}(t_{k+i}), \boldsymbol{c}(t_{k+i})), \\ & \quad \text{for } i = 0, \ldots, N_p - 1 \\ & c_{\min} \le \boldsymbol{c}(t_{k+i}) \le c_{\max},\end{aligned} \tag{45}$$

where $\mathbf{T}_Q, \mathbf{R}$, and $\mathbf{Q}$ represent the weight matrices. After solving for the optimal sequence $\boldsymbol{c}(t_k), \ldots, \boldsymbol{c}(t_{k+N_p-1})$, only the first control action, $\mathbf{c}(t_k)^*$, is applied to the system. The rest of the sequence is discarded, and this entire optimization process is repeated from the next time step t_{k+1}. This formulation is related to DeepONet MPC de Jong et al. (2025), but differs in that we use a one-step DeepONet surrogate $\mathcal{Y}_\theta$ and obtain multi-step predictions by rolling it out recursively over the horizon, rather than using a multi-step DeepONet that predicts the full trajectory in a single forward pass.

Choice of controller. For the voltage optimization task, we use *Direct*, *SBTO* and *Adjoint-method for open loop*. For the heat equation, CN yields a linear PDE, so the finite-horizon problem is a convex QP, which can be solved with *Linear MPC*. For Burgers' equation, the dynamics are nonlinear and CN produces an implicit nonlinear update, so we use *Nonlinear MPC*, with NMPC optimizing over an implicit CN step. For both the heat and burgers' tasks, we use *Adjoint-method for closed loop*, *SBTO* and *DeepONet MPC*.

7.8 MEASURING FEASIBILITY AT INFERENCE TIME

The relaxation in (8) penalizes constraint violations during training but does not guarantee that the predicted solution is feasible at inference time. We therefore evaluate feasibility at inference time by measuring each block of $\boldsymbol{h}'$ in (9) together with the bound constraints. For a predicted trajectory $\{\hat{\boldsymbol{y}}_k\}_{k=0}^{N_t}$ and controls $\{\hat{\boldsymbol{c}}_k\}$, with $\mathcal{A}, \mathcal{B}$ the Crank–Nicolson operators and $\boldsymbol{s}_k$ the source term, we report

$$\boldsymbol{r}_k = \mathcal{A}\hat{\boldsymbol{y}}_{k+1} - (\mathcal{B}\hat{\boldsymbol{y}}_k + \Delta t\, \boldsymbol{s}_k)\,, \tag{46}$$

$$R_{\text{IC}} = \text{RMS}\big(\hat{\boldsymbol{y}}_0 - \boldsymbol{y}_0\big), \tag{47}$$

$$R^{\text{D}}_{\text{BC}} = \text{RMS}_k\big(\hat{y}_k^{(0)},\, \hat{y}_k^{(-1)}\big), \tag{48}$$

$$R^{\text{N}}_{\text{BC}} = \text{RMS}_k\big(\partial_x \hat{y}_k|_{\partial\Omega}\big), \tag{49}$$

$$v_i = \max\big(0,\, u_i - u_{\max},\, u_{\min} - u_i\big), \tag{50}$$

which correspond to the dynamics residual, the initial-condition error, the Dirichlet and Neumann boundary errors, and the bound violation. Here $\boldsymbol{r}_k$ is reported relative to $\|\mathcal{B}\hat{\boldsymbol{y}}_k + \Delta t\, \boldsymbol{s}_k\|$, $\partial_x \hat{y}$ uses a second-order one-sided stencil, and v_i is summarized by conditional mean magnitude and violation rate; the per-benchmark operators are given in Appendix 8. Since the Neumann boundary residual (49) is generally nonzero even for the exact discrete solution due to discretization, each metric is reported together with the corresponding value obtained by evaluating the ground-truth numerical solution.

7.9 EXPERIMENTAL SETTINGS

This section provides the mathematical models and corresponding description of the tasks considered in the Experimental Setting section of the main paper. The optimization problems are formulated with box constraints $[u_{\min}, u_{\max}]$ (or $[c_{\min}, c_{\max}]$ for basis weights) representing physical actuator limits.

7.9.1 OPTIMAL CONTROL OF THE VOLTAGE EQUATION.

The problem of regulating the voltage magnitude in a leaky electrical transmission line, subject to a reaction-diffusion PDE with Neumann conditions is described as follows:

$$\begin{aligned}
\underset{u(x)}{\text{Minimize}} \quad & \int_\Omega (y(T,x) - y_{\text{target}}(x))^2 dx \\
& + \gamma \int_\Omega u(x)^2 dx && \text{(51a)} \\
\text{s.t.} \quad & \frac{\partial y}{\partial t} = D\frac{\partial^2 y}{\partial x^2} - \beta(y - y_{\text{ref}}(x)) + \alpha u(x), && \text{(51b)} \\
& \qquad\qquad \forall (x,t) \in \Omega \times [0,T] \\
& y(0,x) = y_0(x), \forall x \in \Omega && \text{(51c)} \\
& \frac{\partial y}{\partial x}(t,x) = 0, \forall x \in \partial\Omega,\ t \in [0,T] && \text{(51d)} \\
& u_{\min} \le u(x) \le u_{\max}, \forall x \in \Omega. && \text{(51e)}
\end{aligned}$$

Here, $y(t,x) \in \mathbb{R}$ represents the voltage profile across the space-time domain defined by the cartesian product between Ω and time horizon $[0,T]$. The system dynamics, expressed by (51b), model the physical phenomena of diffusion ($D\frac{\partial^2 y}{\partial x^2}$), voltage leakage towards a reference $y_{\text{ref}}(x)$ ($-\beta(y - y_{\text{ref}}(x))$) and the applied control action ($\alpha u(x)$). D, β, and α are all constants representing the diffusion rate, leakage rate, and control gain, respectively. The system is initialized from a state $y_0(x)$ according to (51c) and is constrained by zero-flux Neumann boundary conditions (51d), implying no voltage diffusion across the boundaries of the domain. The control decision $u(x)$ is a space-dependent function representing the externally applied voltage, constrained by actuator limits in (51e). The objective (51a) minimizes a combination of terminal cost and control energy.

Datasets and methods. In the system dynamics (51b), we set the diffusion rate $D = 0.1$, leakage rate $\beta = 1.0$, and control gain $\alpha = 2.0$, with a uniform reference voltage $y_{\text{ref}}(x) = 1.0$. The ground-truth dynamics are simulated using a second-order implicit Crank-Nicolson finite-difference method Crank & Nicolson (1947). To initialize the proposed time-discrete Neural Operator $\mathcal{Y}_\theta$, we generate a dataset by sampling a diverse set of control functions $u(x)$ from a zero-mean Gaussian Random Field (GRF) with a squared-exponential kernel, with mean $m(x)$ and covariance function $k_\ell(x_1, x_2)$ as:

$$m(x) = 0, \quad k_\ell(x_1, x_2) = \sigma^2 \exp\left(-\frac{|x_1 - x_2|_2^2}{2\ell^2}\right), \tag{52}$$

where $\ell > 0$ is the length–scale and $\sigma^2 > 0$ the marginal variance. The corresponding solution $y(t,x)$ is computed using a numerical solver, and then split into 80% training, 10% validation, and 10% test sets.

PDE-OP's model $\mathcal{Y}_\theta$ estimates the voltage profile $y(t,x)$; for $t = T$ this prediction yields $y(T,x)$, which is compared against the target profile $y_{\text{target}}(x)$. PDE-OP's surrogate controller $\mathcal{U}_\omega$ is trained to produce optimal control action $\hat{u}(x)$ while minimizing objective (51a). Our approach is benchmarked against Direct, Adjoint-Method and SBTO, which are described in Appendix 7.2-7.6. Notably, while SBTO is described for closed-loop control problems (see Appendix 7.6), for a comprehensive comparison, we adapt it also to open-loop problems. At inference time, PDE-OP's surrogate controller $\mathcal{U}_\omega$ is tested on three unseen target profiles of different level of complexity: constant ($y_{\text{target}}(x) = p_1$), linear ramp ($y_{\text{target}}(x) = p_1 x + p_2$), and sine wave ($y_{\text{target}}(x) = p_1 + p_2(\sin f_0 x + p_3)$). For each target, the control $\hat{u}(x)$ is generated in a single forward pass.

7.9.2 Optimal Control of the 1D Heat Equation.

This task involves controlling the evolution of temperature in a one-dimensional domain described by a reaction–diffusion PDE. Given an initial profile, the controller aims to reach a prescribed target state with minimal control effort. The problem is formulated as:

$$\underset{\boldsymbol{c}(t)_{t\in[0,T]}}{\text{Minimize}} \quad \mathcal{L} = \mathcal{L}_{\text{terminal}} + \lambda\mathcal{L}_{\text{running}} + \gamma\mathcal{L}_{\text{effort}} \tag{53a}$$

$$\text{s.t.} \quad \frac{\partial y}{\partial t} = D\frac{\partial^2 y}{\partial x^2} - \beta(y - y_{\text{ref}}(x)) + \alpha u(t,x), \quad \forall (x,t) \in \Omega \times [0,T] \tag{53b}$$

$$u(t,x) = \boldsymbol{c}(t)^\top \boldsymbol{\phi}(x),\ \forall (x,t) \in \Omega \times [0,T] \tag{53c}$$

$$y(0,x) = y_0(x), \quad \forall x \in \Omega \tag{53d}$$

$$\frac{\partial y}{\partial x}(t,x) = 0, \quad \forall x \in \partial\Omega,\ t \in [0,T] \tag{53e}$$

$$c_{\min} \le \boldsymbol{c}(t) \le c_{\max}, \quad \forall\, t \in [0,T]. \tag{53f}$$

The loss components in (53a) are defined as follows.

$$\mathcal{L}_{\text{terminal}} := \int_\Omega |y(T,x) - y_{\text{target}}(x)|^2 \, dx, \tag{54a}$$

$$\mathcal{L}_{\text{running}} := \int_0^T \int_\Omega |y(t,x) - y_{\text{target}}(x)|^2 \, dx dt, \tag{54b}$$

$$\mathcal{L}_{\text{effort}} := \int_0^T ||\boldsymbol{c}(t)||_2^2 \, dt. \tag{54c}$$

Here $y(t,x)$ represents the temperature profile. The objective (53a) balances three terms: a terminal loss for final state accuracy, a running loss for temperature tracking, and an effort loss for control efficiency, described by (54a)–(54c).

Datasets and methods. The solutions of (53b) are generated using a Crank-Nicolson method, which deals with time-varying source term by averaging the control input over each time step. To train our models, we generate a dataset of diverse trajectories. First, smooth, time-varying control weights ($\boldsymbol{c}(t)$) are generated by sampling each of the M weight trajectories from a GRF over the time domain. The full spatial-temporal control field $u(t,x)$ is then reconstructed from these weights constrained by (53f) using a set of fixed basis functions, as defined in (53c). For each generated control field, the full PDE is solved to generate a ground-truth temperature evolution $y(t,x)$. Each complete simulation yields a training sample, consisting of the control input weight trajectory $\{\boldsymbol{c}(t_k)\}$, and the corresponding state trajectory $\{y(t_k,x)\}$ evaluated at a set of fixed sensor locations. The dataset is then split into 80% training, 10% validation and 10% test.

Our framework uses two neural networks. First, a discrete-time neural operator, $\mathcal{Y}_\theta$, is trained as a surrogate model described in the *Discrete-time Neural Operator* section of the main text. It act as a one-step time-advancement operator, learning the mapping $(y(t_k,x), \boldsymbol{c}(t_k)) \to y(t_{k+1},x)$. Second, PDE-OP's surrogate controller $\mathcal{U}_\omega$ is a recurrent neural network that acts as a sequential decision-maker. At each time step t_k, it takes the current state $y(t_k,x)$ and the final target $y_{\text{target}}(x)$ to produce the control weights ($\boldsymbol{c}(t_k)$). The controller is trained end-to-end by unrolling the trajectory using the surrogate $\mathcal{Y}_\theta$. We benchmark our approach against a highly-tuned Adjoint-Method, Linear MPC(LMPC), SBTO, and DeepONet MPC. Further implementation details are in Appendix 7.4-7.7.

At inference, we evaluate PDE-OP model on six unseen target profiles: constant($y_{\text{target}}(x) = p_1$), linear ramp($y_{\text{target}}(x) = p_1 x + p_2$), sine wave($y_{\text{target}}(x) = p_1 + p_2(\sin f_0 x + p_3)$, step(discontinuous), high-frequency($y_{\text{target}}(x) = p_1 \sin(f_1 x) + p_2 \sin(f_2 x)$), and complex-gaussian (mixed of gaussian profiles).

7.9.3 Optimal Control of the 2D Heat Equation.

To consider a more realistic and complex scenario, we extend the temperature control task introduced in the previous subsection to a two-dimensional spatial domain $\Omega = [0, L_x] \times [0, L_y] \subset \mathbb{R}^2$. Similar

to the 1D case, the controller seeks to drive the system from an initial temperature profile to a target state using minimal control effort. The control input is distributed spatially via a set of 2D basis functions. The optimization problem is formulated as follows:

$$\underset{\boldsymbol{c}(t)_{t\in[0,T]}}{\text{Minimize}} \quad \mathcal{L} = \mathcal{L}_{\text{terminal}} + \lambda\mathcal{L}_{\text{running}} + \gamma\mathcal{L}_{\text{effort}} \tag{55a}$$

$$\text{s.t. } \frac{\partial z}{\partial t} = D\left(\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2}\right) - \beta(z - z_{\text{ref}}) + \alpha u(t,\mathbf{x}), \quad \forall(\mathbf{x},t) \in \Omega \times [0,T] \tag{55b}$$

$$u(t,\mathbf{x}) = \boldsymbol{c}(t)^\top \boldsymbol{\Phi}(\mathbf{x}),\ \forall(\mathbf{x},t) \in \Omega \times [0,T] \tag{55c}$$

$$z(0,\mathbf{x}) = z_0(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \tag{55d}$$

$$\nabla z \cdot \mathbf{n} = 0, \quad \forall \mathbf{x} \in \partial\Omega,\ t \in [0,T] \tag{55e}$$

$$c_{\min} \le \boldsymbol{c}(t) \le c_{\max}, \quad \forall\, t \in [0,T]. \tag{55f}$$

Here, $\mathbf{x} = (x, y)$ denotes the spatial coordinates. The boundary condition (55e) represents homogeneous Neumann conditions (which correspond to insulated boundaries), where $\mathbf{n}$ is the outward normal vector. The control input $u(t,\mathbf{x})$ is represented as a linear combination of M basis functions with basis weights $\boldsymbol{c}(t) \in \mathbb{R}^M$, where $M = N_{b,x} \times N_{b,y}$. The spatial basis $\boldsymbol{\Phi}(\mathbf{x})$ is defined as the product of cosine functions: $\phi_{ij}(x,y) = \cos(\frac{i\pi x}{L_x})\cos(\frac{j\pi y}{L_y})$.

The loss components are defined over the 2D domain:

$$\mathcal{L}_{\text{terminal}} := \int_\Omega |z(T,\mathbf{x}) - z_{\text{target}}(\mathbf{x})|^2\, d\mathbf{x}, \tag{56a}$$

$$\mathcal{L}_{\text{running}} := \int_0^T \int_\Omega |z(t,\mathbf{x}) - z_{\text{target}}(\mathbf{x})|^2\, d\mathbf{x}dt, \tag{56b}$$

$$\mathcal{L}_{\text{effort}} := \int_0^T ||\boldsymbol{c}(t)||_2^2\, dt. \tag{56c}$$

Datasets and methods. The system in equation (55b) is discretized on a tensor-product grid using a 2D Crank–Nicolson scheme. To efficiently handle spatial derivatives, the discrete Laplacian is constructed via the Kronecker sum of 1D operators. To generate the training dataset we sample diverse control trajectories where the weights $\boldsymbol{c}(t)$ are drawn from a GRF with varying temporal length scales l. The resulting control field $u(t,\mathbf{x})$ is computed via Equation (55c), and the PDE is solved to obtain the corresponding temperature evolution $z(t,\mathbf{x})$. The solution is then interpolated onto a fixed sensor grid of size $N_{s,x} \times N_{s,y}$ and the spatial fields are flattened into vectors of length $N_{s,x}N_{s,y}$ to be used as inputs for the neural operator. Each training sample consists of a control weight trajectory $\boldsymbol{c}(t)$ and the corresponding flattened state trajectory $z(t,\mathbf{x})$. The dataset is split into 80% training, 10% validation, and 10% test sets.

Following the 1D setup, we use two neural networks: a discrete-time neural operator $\mathcal{Y}_\theta$ adapted for 2D inputs (via a DeepONet with 2D trunk input), and a recurrent surrogate controller $\mathcal{U}_\omega$ that maps the flattened sensor data to control weight outputs for the 2D basis functions. We benchmark our approach against the Adjoint-Method, Linear MPC(LMPC), SBTO and DeepONet MPC. A description of each of these methods can be found in Appendix 7.4-7.7.

At test time, we evaluate the PDE-OP model on two unseen target profiles: a *Gaussian peak* ($z_{\text{target}}(\mathbf{x}) = A\exp(-\frac{(x-c_x)^2}{2w_x^2} - \frac{(y-c_y)^2}{2w_y^2})$), a *Sine wave* ($z_{\text{target}}(\mathbf{x}) = \mu + A\sin(2\pi x/L_x)\sin(2\pi y/L_y)$).

7.9.4 Optimal Control of the 1D Burgers' Equation.

To test the ability of PDE-OP to handle nonlinear dynamics, we address the control of the 1D viscous Burgers' equation with Dirichlet boundary condition. The control input and external force $u(t,x)$, is parameterized identically to the heat equation task, using time-varying weights $(\boldsymbol{c}(t_k))$ for a set of basis functions. The optimization goal is to find the optimal trajectory of these weights subject to Burgers' equation (57b), initial (57d) and Dirchlet boundary condition (57e) and box constraints

(57f).

$$\underset{\boldsymbol{c}(t)_{t\in[0,T]}}{\text{Minimize}} \quad \mathcal{L} = \mathcal{L}_{\text{terminal}} + \lambda\mathcal{L}_{\text{running}} + \gamma\mathcal{L}_{\text{effort}} \tag{57a}$$

$$\text{s.t. } \frac{\partial y}{\partial t} + y\frac{\partial y}{\partial x} = \nu\frac{\partial^2 y}{\partial x^2} + u(t,x), \quad \forall (x,t) \in \Omega \times [0,T] \tag{57b}$$

$$u(t,x) = \boldsymbol{c}(t)^\top \boldsymbol{\phi}(x), \quad \forall (x,t) \in \Omega \times [0,T] \tag{57c}$$

$$y(0,x) = y_0(x), \qquad \forall x \in \Omega \tag{57d}$$

$$y(t,x) = 0, \qquad \forall x \in \partial\Omega,\ t \in [0,T] \tag{57e}$$

$$c_{\text{min}} \le \boldsymbol{c}(t) \le c_{\text{max}}, \qquad \forall\, t \in [0,T]. \tag{57f}$$

Here $y(t,x)$ is the velocity field and ν represents the viscosity parameter. The PDE in (57b) contains convective nonlinearity $(y\frac{\partial y}{\partial x})$ which poses a significant control challenge not present in the linear heat equation. The objective retains the same structure, balancing final state accuracy, path tracking, and control efficiency.

Datasets and methods. The ground-truth dynamics for the nonlinear Burgers' equation are simulated using a high-fidelity numerical solver. The dataset generation process mirrors that of the heat equation: time-varying weights $\boldsymbol{c}(t)$ are sampled from a GRF, and the force field $u(t,x)$ is reconstructed using spatial basis functions, and the PDE is solved to generate trajectories. Due to the system's high nonlinearity, we benchmark our method against a much more general and computationally expensive Nonlinear MPC (NMPC) and Adjoint-Sensitivity Method. Further implementation details are in Appendix 7.4 - 7.5.

At inference, we test the PDE-OP model on three unseen target profiles: constant ($y_{\text{target}}(x) = p_1$), parabolic ($y_{\text{target}}(x) = p_1 x(1-x)$) and sine wave ($y_{\text{target}}(x) = p_1(\sin f_0 x + p_2)$). We do not use a linear target profile, because of the Dirichlet BCs.

7.9.5 Optimal Control of the 1D Burgers' Equation with Spatially-Varying Viscosity.

To evaluate the ability of PDE-OP to generalize across heterogeneous physical systems, we address the control of the 1D viscous Burgers' equation with a spatially-varying viscosity parameter $\nu(x)$. This introduces a parametric complexity where the controller must adapt to the specific material properties of the domain. The optimization goal remains to find the optimal trajectory of basis function weights $\boldsymbol{c}(t)$ subject to the modified dynamics and Dirichlet boundary conditions:

$$\underset{\boldsymbol{c}(t)_{t\in[0,T]}}{\text{Minimize}} \quad \mathcal{L} = \mathcal{L}_{\text{terminal}} + \lambda\mathcal{L}_{\text{running}} + \gamma\mathcal{L}_{\text{effort}} \tag{58a}$$

$$\text{s.t. } \frac{\partial y}{\partial t} + y\frac{\partial y}{\partial x} = \frac{\partial}{\partial x}\left(\nu(x)\frac{\partial y}{\partial x}\right) + u(t,x), \quad \forall (x,t) \in \Omega \times [0,T] \tag{58b}$$

$$u(t,x) = \boldsymbol{c}(t)^\top \boldsymbol{\phi}(x), \quad \forall (x,t) \in \Omega \times [0,T] \tag{58c}$$

$$y(0,x) = y_0(x), \qquad \forall x \in \Omega \tag{58d}$$

$$y(t,x) = 0, \qquad \forall x \in \partial\Omega,\ t \in [0,T] \tag{58e}$$

$$c_{\text{min}} \le \boldsymbol{c}(t) \le c_{\text{max}}, \qquad \forall\, t \in [0,T]. \tag{58f}$$

Here $y(t,x)$ is the velocity field. The diffusive term in (58b) is now governed by $\nu(x)$, which varies across the domain Ω. This setup presents a dual challenge: the controller must handle the convective nonlinearity $(y\partial_x y)$ while simultaneously compensating for regions of high and low dissipation defined by $\nu(x)$.

Datasets and methods. The ground-truth dynamics are generated using a numerical Crank-Nicolson solver on a staggered grid, which ensures accurate flux conservation under varying viscosity conditions. To construct the dataset, we randomize both the control inputs and the physical parameters: control weights $\boldsymbol{c}(t)$ are sampled from a temporal GRF, while the spatial viscosity profile $\nu(x)$ is drawn from a spatial GRF and mapped to a bounded positive range $[\nu_{\min}, \nu_{\max}]$. To account for this variability, both the neural operator and the surrogate controller are designed to

be *parameter-conditioned*, which is obtained by concatenating the discretized viscosity profile $\boldsymbol{\nu}$ to their respective inputs.

We benchmark PDE-OP against the classical method Adjoint-Sensitivity Method, which assumes full knowledge of the system dynamics. Additionally, we compare against two learning-based baselines: SBTO, which optimizes control inputs using a pretrained operator model, and DeepONet MPC, which uses the surrogate model for look-ahead receding-horizon planning.

At inference time, we evaluate the models on six target profiles to assess generalization performance: (i) *Sine Mixture* ($y_{\text{target}} \propto 0.4\sin(2\pi x) + 0.4\sin(4\pi x)$); (ii) *Parabola* ($y_{\text{target}} \propto x(L-x)$); (iii) *Step Function* ($y_{\text{target}} = 0.6$ for $x \in [0.3L, 0.7L]$); (iv) *High-Frequency* ($y_{\text{target}} \propto 0.5\sin(2\pi x) + 0.2\sin(12\pi x)$); (v) *Complex Gaussian*, a stochastic superposition of 5 random Gaussian modes; and (vi) *Zero* ($y_{\text{target}} = 0$). All evaluations are conducted under randomized viscosity profiles $\nu(x)$ to test the robustness of the controllers to variations in the physical environment.

7.9.6 Optimal Control of the 1D Heat Equation (Battery Thermal Management).

Here the task consists of controlling the thermal profile of a Lithium-Ion battery module, modeled as a 1D heat equation as described in Section 7.9.2. This task is particularly important in high-performance energy systems, as maintaining an optimal temperature distribution is critical for safety and efficiency.

Dataset and Physical Parameters.

To reflect real-world energy system behavior, we adopt the PDE parameters from Liu et al. (2019). Specifically, the thermal diffusivity D is derived from the material properties of a commercial Lithium-Ion polymer battery cell (EiG ePLB-C020), as experimentally validated in that study. Given the density $\rho = 2092\,\text{kg/m}^3$, specific heat capacity $C_p = 678\,\text{J/(kg}\cdot\text{K)}$, and thermal conductivity $k \approx 18.4\,\text{W/(m}\cdot\text{K)}$, the physical thermal diffusivity D_{phys} is calculated as:

$$\begin{aligned} D_{\text{phys}} = \frac{k}{\rho \cdot C_p} &= \frac{18.4}{2092 \cdot 678} \\ &\approx 1.3 \times 10^{-5}\,\text{m}^2/\text{s} = 0.13\,\text{cm}^2/\text{s}. \end{aligned} \tag{59}$$

The PDE solutions used to initialize the neural surrogate of PDE-OP and the neural baselines are generated using a Crank-Nicolson method. In particular, we generate a dataset of diverse trajectories by sampling smooth, time-varying control weights $c(t)$ from a Gaussian Random Field. The dataset is divided into 80% training, 10% validation, and 10% test. We compare our approach against the Adjoint-Method, Linear MPC, SBTO, and DeepONet MPC.

7.10 Experiment Results

In this section, we first evaluate the ability of the time-discrete neural operator introduced in the *Discrete-time Neural Operator* section of the main text in capturing the system dynamics.
Secondly, we evaluate the runtime and terminal tracking error (MSE) achieved by each method on different target profiles, and report these metrics in Tables 5- 13. All methods use the same spatial/temporal discretization and solver settings as described in Appendix 10.

7.10.1 Capturing the System Dynamics with PDE-OP's time-discrete Neural Operator.

Figure 5 shows the predicted voltage (central sub-figure of Fig. 5) of PDE-OP's dynamic component $\mathcal{Y}_\theta$ at test time, which is compared to the solution (left sub-figure) obtained with a numerical PDE-solver. Figure 6 shows the 2D temperature distribution estimated by PDE-OP's dynamic component $\mathcal{Y}_\theta$ at test time across different time steps ($t = 0.25, 0.50, 1.00$), compared to the numerical PDE solver. The predicted temperature field closely tracks the ground-truth dynamics across the spatial domain, with absolute error remaining on the order of 10^{-2} throughout the rollout. Figure 7 shows the system dynamics (central sub-figure of Fig. 7) estimated by PDE-OP's dynamic component $\mathcal{Y}_\theta$ at test time, which is compared to the solution (left sub-figure) obtained with a numerical PDE-solver. The figure shows that the predicted velocity field aligns closely with the numerical PDE-solver

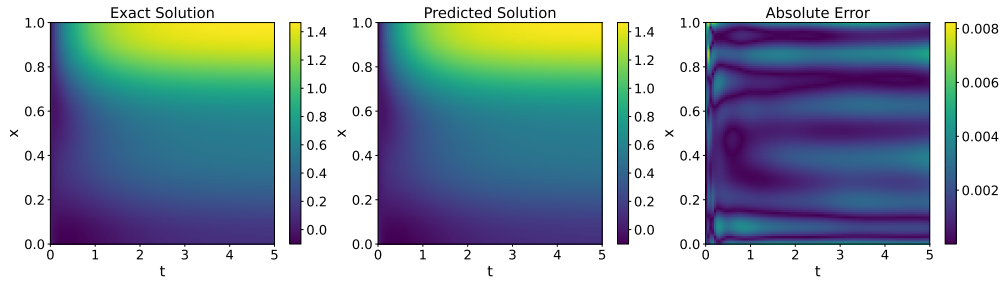

Figure 5: Voltage Equation. PDE-OP's dynamic predictor $\mathcal{Y}_\theta$ solution estimate at test time: numerical solution (left), model prediction (center), and absolute error (right).

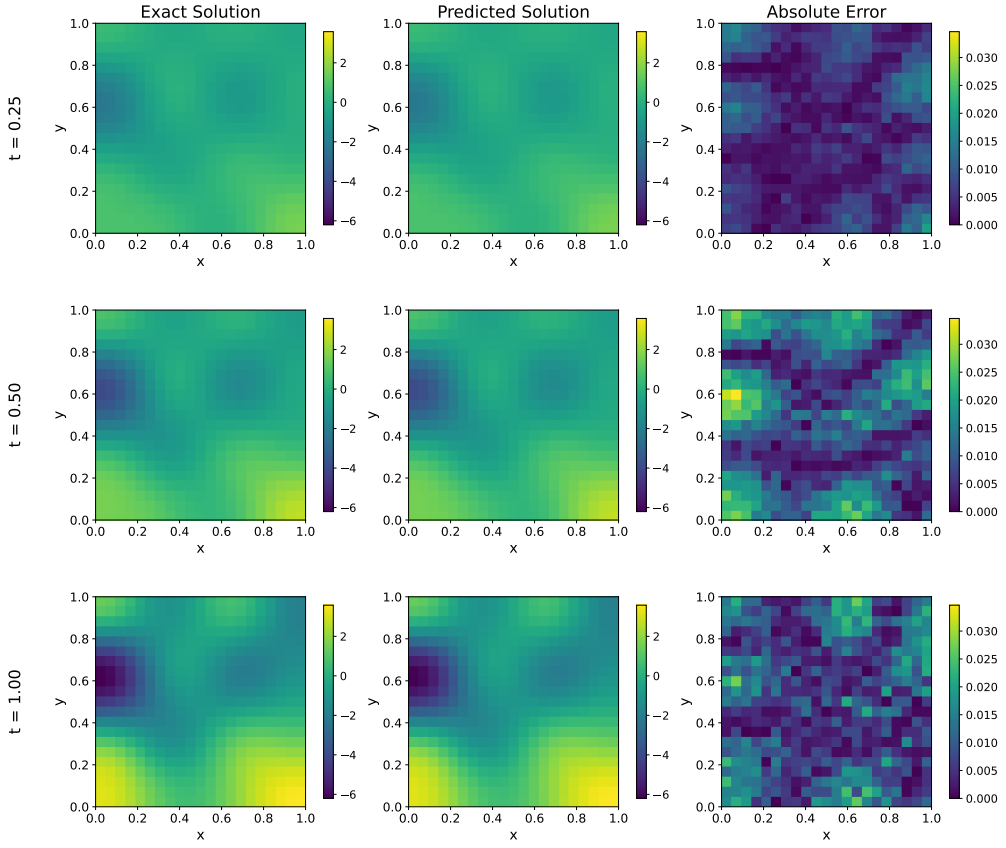

Figure 6: 2D Heat Equation. PDE-OP's dynamic predictor $\mathcal{Y}_\theta$ solution estimate at test time across temporal snapshots ($t = 0.25, 0.50, 1.00$ from top to bottom): numerical ground truth (left), model prediction (center), and absolute error (right).

solution; with absolute error on the order of 10^{-3} across the space–time domain, demonstrating the ability of PDE-OP's dynamic predictor to accurately capture Burgers' dynamics.

7.10.2 Optimal Control of the Voltage Equation.

We show other targets $y_{\text{target}}(x) = p_1 x + p_2$: (i) *constant*: $(p_1, p_2) = (0, 1)$ so $y_{\text{target}}(x) = 1$; (ii) *ramp*: $(p_1, p_2) = (1, 0.5)$ so $y_{\text{target}}(x) = x + 0.5$. (Equivalently, $y_{\text{target}}(x) = p_1 x + p_2$ with $(p_1, p_2) = (0, 1)$ and $(1, 0.5)$, respectively.) Since a ramp target profile is infeasible under zero-flux (Neumann) boundaries, all methods can only produce approximations.

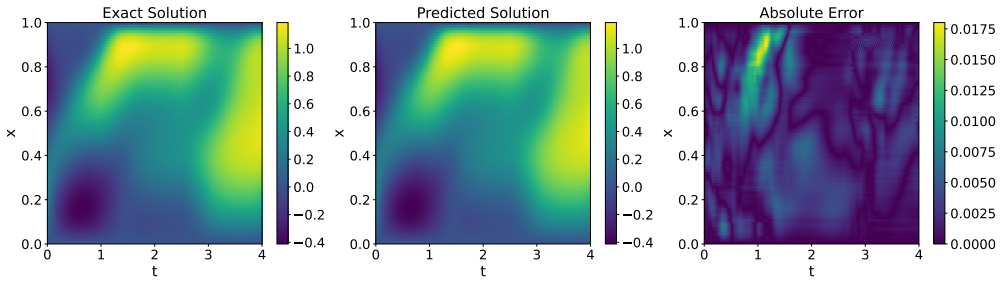

Figure 7: 1D Burgers' Equation. PDE-OP's dynamic predictor $\mathcal{Y}_\theta$ solution estimate at test time: numerical solution (left), model prediction (center), and absolute error (right).

Table 5: Voltage Equation. For each method and target profile, we report runtime (s) and MSE.

Methods	**Constant**		**Ramp**	
	Runtime	**Accuracy**	**Runtime**	**Accuracy**
Direct (Finite Diff.)	2.092s	4.7441e-11	11.113s	5.0784e-05
Adjoint-Method	0.049s	**4.6824e-11**	0.644s	**3.6766e-05**
SBTO	0.508s	1.0701e-08	0.510s	3.1697e-04
PDE-OP (ours)	**0.037s**	3.6866e-06	**0.050s**	2.4211e-04

Results. *Constant target* $y_{target}(x) = 1$. PDE-OP is the fastest method on this task, requiring only $\mathbf{0.037}$ s while achieving an MSE of 3.69×10^{-6}, corresponding to speedups of approximately $56\times$ over the Direct method and $14\times$ over SBTO. The Adjoint method attains the lowest error (MSE 4.68×10^{-11}) in 0.049 s, while SBTO achieves an MSE of 1.07×10^{-8} in 0.508 s. The Direct method obtains a comparable MSE of 4.74×10^{-11} but requires 2.092 s. On this target, however, the runtime advantage over the Adjoint method essentially disappears ($1.1\times$), since a constant profile is the easiest instance of this benchmark and the Adjoint solver converges in only a handful of iterations. Consequently, PDE-OP's fixed inference cost becomes comparable to that of a full numerical solve. Moreover, the Adjoint and Direct methods both operate at the numerical accuracy floor of this problem, making the difference between their errors insignificant, while PDE-OP still achieves a very small tracking error (3.69×10^{-6}) that remains negligible for this task.

Ramp target $y_{target}(x) = x + 0.5$. PDE-OP runs in $\mathbf{0.050}$ s, approximately $222\times$ faster than the Direct method, $12.9\times$ faster than the Adjoint method, and $10.2\times$ faster than SBTO, while achieving an MSE of 2.42×10^{-4}. The Adjoint method attains the lowest error (MSE 3.68×10^{-5}) in 0.644 s, whereas SBTO achieves an MSE of 3.17×10^{-4} in 0.510 s. Although PDE-OP does not match the accuracy of the Adjoint method, it substantially outperforms SBTO and remains on the same order of magnitude as the numerical baselines while providing over an order-of-magnitude reduction in runtime. Unlike the constant target, this profile is infeasible under zero-flux boundary conditions, requiring substantially more optimization iterations for the numerical solvers. Consequently, PDE-OP's amortized inference retains a clear computational advantage.

7.10.3 Optimal Control of the 2D Heat Equation

For this task, we evaluate PDE-OP and the baseline methods on two distinct target profiles defined in the spatial domain $\Omega = [0, L_x] \times [0, L_y]$: (i) *Gaussian Peak*, a single centered mode $z_{\text{target}}(\mathbf{x}) = 1.5 \exp\left(-\frac{(x-c_x)^2}{2w_x^2} - \frac{(y-c_y)^2}{2w_y^2}\right)$; (ii) *Sine Wave*, a spatially periodic profile $z_{\text{target}}(\mathbf{x}) = 0.8 + 0.7 \sin(2\pi x/L_x) \sin(2\pi y/L_y)$. We compare the proposed PDE-OP model against the Adjoint-Sensitivity Method, LMPC and two neural baselines: Surrogate-Based Test-Time Optimization (SBTO) and DeepONet MPC as shown in Figure 4 and in Table 6.

Results. Given a Gaussian target, classical methods (LMPC and Adjoint) achieve the lowest MSE ($\mathbf{6.62 \times 10^{-4}}$), though LMPC is computationally prohibitive for real-time applications (runs in

Table 6: 2D Heat Equation Task. We report runtime (s) and MSE for both target types.

Methods	Gaussian Peak		Sine Wave	
	Runtime	Accuracy	Runtime	Accuracy
LMPC	48.481s	**0.00066**	67.258s	**0.00712**
Adjoint-Method	6.328s	**0.00066**	6.349s	**0.00712**
SBTO	16.049s	0.00070	15.483s	0.00724
DeepONet MPC	17.878s	**0.00066**	17.888s	0.01868
PDE-OP (ours)	**0.031s**	0.00081	**0.031s**	0.00721

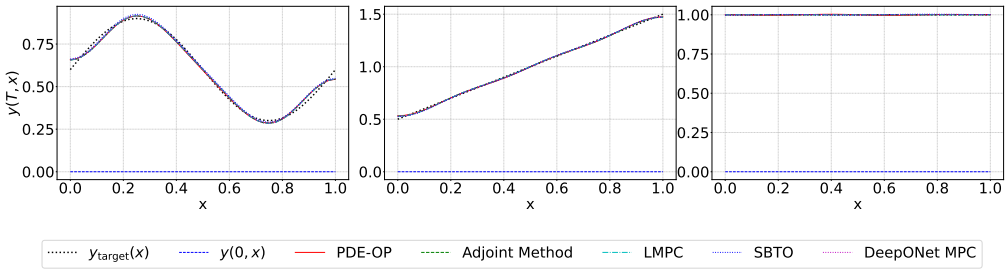

Figure 8: 1D Heat Equation. Comparison of PDE-OP and other methods for three target profiles: $y_{\text{target}}(x) = 0.6 + 0.3(\sin(2\pi x))$(left), $y_{\text{target}}(x) = x + 0.5$(center), and $y_{\text{target}}(x) = 1$(right).

48.5 s). The neural baselines, SBTO and DeepONet MPC, achieve competitive accuracy but require iterative optimization at test time (≈ 17 s). PDE-OP matches their accuracy level (8.1×10^{-4}) but runs in only $\mathbf{0.03}$ s, approximately $\mathbf{518}\times$ faster than SBTO and $\mathbf{204}\times$ faster than the Adjoint method.

7.10.4 Optimal Control of the 1D Burgers' Equation

For the 1D viscous Burgers equation with Dirichlet boundaries $y(t, 0) = y(t, L) = 0$, admissible steady targets must satisfy these conditions. We therefore consider (i) a *zero* (constant) target $y_{\text{target}}(x) = 0$, and (ii) a *parabolic* target $y_{\text{target}}(x) = p_1\, x(1 - x)$ (we use $p_1 = 2.0$ in experiments). We compare our method with NMPC and time-varying Adjoint baseline.

Table 7: Burgers' Equation. For each method and target profile we report runtime (s) and MSE.

Methods	Constant		Parabola	
	Runtime	Accuracy	Runtime	Accuracy
NMPC	132.243s	**5.6760e-14**	671.033s	0.00006
Adjoint-Method	18.131s	5.8269e-11	103.969s	1.2944e-04
SBTO	66.074s	1.9443e-07	66.099s	7.9198e-6
DeepONet MPC	98.296s	3.3844e-6	98.208s	**1.8547e-6**
PDE-OP (ours)	**0.176s**	4.2253e-05	**0.176s**	3.1101e-03

Results. *Constant target* $y_{target}(x) = 0$. PDE-OP is the fastest method, requiring only $\mathbf{0.176}$ s, corresponding to speedups of approximately 7.5×10^2 over NMPC and 1.03×10^2 over the Adjoint method, while achieving an MSE of 4.2×10^{-5}. NMPC attains the lowest error (MSE 5.68×10^{-14}) but requires 132.243 s. The learning-based baselines, SBTO and DeepONet MPC, achieve MSEs of 1.94×10^{-7} and 3.38×10^{-6}, respectively, while requiring 66.074 s and 98.296 s. The Adjoint method runs in 18.131 s with an MSE of 5.83×10^{-11}. Although PDE-OP does not reach the numerical accuracy floor achieved by the optimization-based methods, it still produces a very small tracking error while reducing inference time by two to three orders of magnitude.

Parabolic target $y_{target}(x) = 2x(1-x)$. PDE-OP is the fastest method, requiring only $\mathbf{0.176}$ s, corresponding to speedups of approximately 3.8×10^3 over NMPC and 5.6×10^2 over DeepONet MPC, while achieving an MSE of 3.1×10^{-3}. DeepONet MPC attains the lowest error (MSE 1.85×10^{-6}), followed by SBTO (7.92×10^{-6}), both outperforming NMPC (6.0×10^{-5}). However, their runtimes remain substantially higher at 98.208 s and 66.099 s, respectively.
Takeaway. For nonlinear Burgers dynamics with Dirichlet BCs, while DeepONet MPC and SBTO can provide superior accuracy on complex profiles, PDE-OP delivers three to four orders of magnitude lower runtime with modest accuracy loss, making it uniquely suitable for real-time applications.

7.10.5 Optimal Control of the 1D Burgers' Equation with Spatially Varying Viscosity

To evaluate the robustness of PDE-OP and the baseline methods against heterogeneous physical parameters and nonlinear dynamics, we test these methods on the 1D Burgers' equation control task (which is formalized in Problem (58)), where the viscosity profile $\nu(x)$ is a function of the space variable x. Due to its high-computational cost, NMPC is not included as a baseline in our evaluation.

Each method is evaluated on the following target profiles $y_{\text{target}}(x)$:

1. **Standard Targets:** Smooth profiles satisfying the boundary conditions.
   - *Sine Mixture*: $y_{\text{target}}(x) = 0.4\sin(2\pi x/L) + 0.4\sin(4\pi x/L)$.
   - *Parabola*: $y_{\text{target}}(x) = 2.0x(L-x)/L^2$.
   - *Zero constant*: $y_{\text{target}}(x) = 0$ (stabilization task).
2. **Advanced Targets:** Discontinuous, high-frequency, or stochastic profiles.
   - *Step*: $y_{\text{target}}(x) = 0.6$ for $x \in [0.3L, 0.7L]$, else 0.
   - *High-Frequency*: $y_{\text{target}}(x) = 0.5\sin(2\pi x/L) + 0.2\sin(12\pi x/L)$.
   - *Complex Gaussian*: A stochastic superposition of five random Gaussian modes, $y_{\text{target}}(x) = \sum_{i=1}^{5} A_i \exp(\frac{-(x-\mu_i)^2}{2\sigma_i^2})$.

We compare PDE-OP against the Adjoint Method, SBTO and DeepONet MPC as shown in Figures 9-10.

Table 8: Burgers' Equation with varying coefficients. For each method and (common) target profile we report runtime (s) and MSE.

	Sine Mixture		**Parabola**		**Zero**	
Method	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**
Adjoint Method	194.271s	0.03440	143.632s	0.00034	41.93s	**5.7012e-8**
SBTO	122.242s	0.00990	121.589s	0.00021	121.581s	0.00020
DeepONet MPC	200.005s	0.03550	199.594s	**0.00016**	199.967s	0.00019
PDE-OP (Ours)	**0.49s**	**0.00918**	**0.356s**	0.00051	**0.342s**	0.00124

Table 9: 1D Burgers' Equation with varying coefficients. For each method and (complex) target profile we report runtime (s) and MSE.

	Step		**High Frequency**		**Complex Gaussian**	
Method	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**
Adjoint Method	177.382s	0.01400	159.489s	0.02080	147.412s	0.11900
SBTO	120.641s	**0.01070**	121.745s	0.04240	121.35s	0.10100
DeepONet MPC	200.012s	0.01470	199.885s	**0.02050**	199.791s	0.11500
PDE-OP (Ours)	**0.342s**	0.01510	**0.343s**	0.02230	**0.342s**	**0.09840**

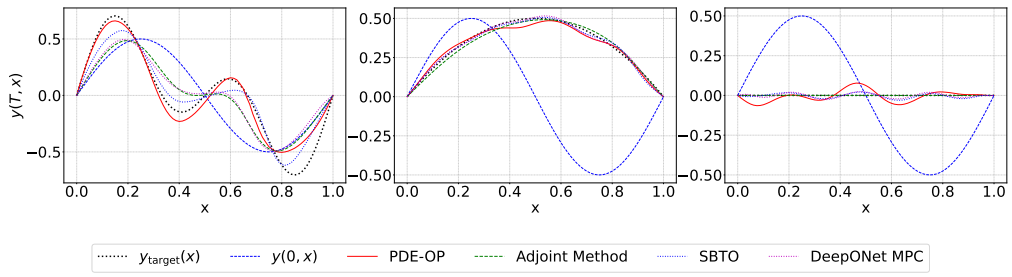

Figure 9: 1D Burgers' Equation with spatially varying coefficient. Comparison between PDE-OP and each baseline method's solution on three different target profile: $y_{\text{target}}(x) = 0.4(\sin(2\pi x)) + 0.4(\sin(4\pi x))$ (left), $y_{\text{target}}(x) = 2x(1-x)$ (center), and $y_{\text{target}}(x) = 0$ (right).

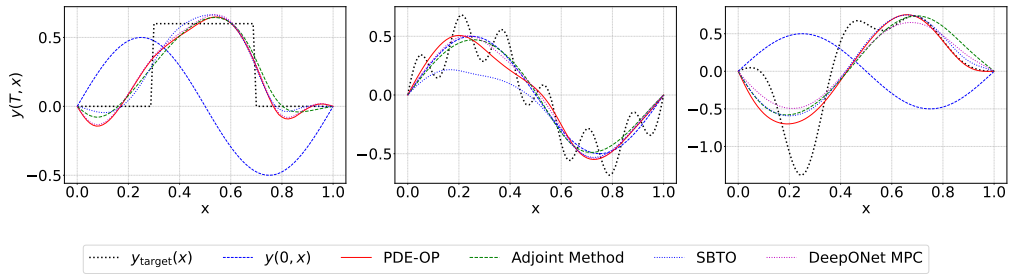

Figure 10: 1D Burgers' Equation with spatially varying coefficient. Comparison of PDE-OP and baselines for advanced target profiles: *Step* (discontinuous)(left), *High-Frequency* ($y_{\text{target}}(x) = 0.5\sin(2\pi x) + 0.2\sin(12\pi x)$)(center), and *Complex Gaussian* (stochastic multi-modal profile)(right).

Results. *Standard Targets.* As shown in Table 8 and Figure 9, PDE-OP achieves the lowest error on the Sine Mixture target (MSE $\mathbf{9.18 \times 10^{-3}}$), significantly outperforming the Adjoint Method (3.44×10^{-2}). While the Adjoint method achieves near-machine precision on the trivial Zero target, PDE-OP maintains competitive accuracy across all target profiles while running approximately 120×–340× faster than the fastest baseline, depending on the target.

Advanced Targets. Table 9 reports the inference time of PDE-OP and the baseline methods, along with the corresponding MSE between the desired target and the final state trajectory, across different advanced target profiles. For a *Complex Gaussian* target, PDE-OP achieves the best accuracy (MSE $\mathbf{9.84 \times 10^{-2}}$), outperforming all the baseline methods as depicted in Figure 10. For the Step and High-Frequency targets, PDE-OP matches the accuracy of the Adjoint method and DeepONet MPC (MSE $\approx 10^{-2}$) but requires only $\mathbf{0.34}$ seconds compared to $> 120 - 200$ seconds for the baselines. SBTO struggles with the High-Frequency target, likely due to convergence to local minima.

Takeaway. For the nonlinear Burgers' equation with heterogeneous viscosity, PDE-OP delivers robust control performance across diverse target geometries. It matches or exceeds the accuracy of computationally expensive iterative optimizers on complex tasks while offering three orders of magnitude acceleration in inference speed.

7.10.6 Optimal Control of the 1D Heat Equation

For the 1D heat equation, we evaluate the control performance on two categories of target profiles $y_{\text{target}}(x)$. **Standard Targets:** (i) *Constant* ($y_{\text{target}} = 1$); (ii) *Ramp* ($y_{\text{target}} = x + 0.5$); and (iii) *Sine* ($y_{\text{target}} \propto 0.6 + 0.3\sin(2\pi x)$). **Advanced Targets:** (i) *Step* (discontinuous profile); (ii) *High-Frequency* (superposition of low and high frequency sine waves); and (iii) *Complex Gaussian* (stochastic mixture of Gaussian modes). We compare the proposed PDE-OP model against classical baselines (Adjoint Method, LMPC) and neural optimization baselines (SBTO, DeepONet MPC) as shown in Figures 11 and 8.

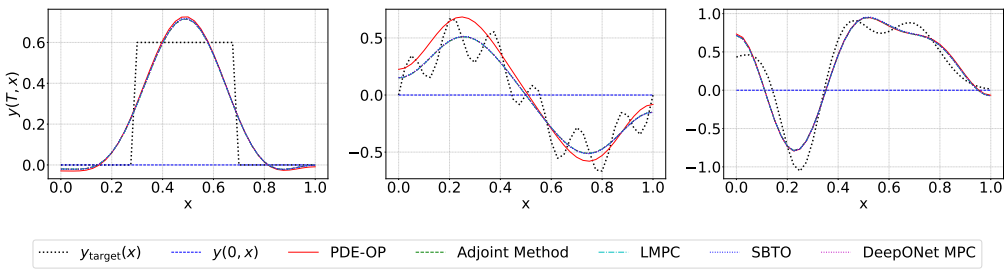

Figure 11: 1D Heat Equation. Comparison of PDE-OP and baselines for advanced target profiles: *Step* (discontinuous)(left), *High-Frequency* ($y_{\text{target}}(x) = 0.5\sin(2\pi x) + 0.2\sin(12\pi x)$)(center), and *Complex Gaussian* (stochastic multi-modal profile)(right).

Table 10: 1D Heat Equation. For each method and standard target profile, we report runtime (s) and MSE.

	Constant		Ramp	
Method	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**
LMPC	0.443s	**2.3289e-14**	0.562s	**0.00008**
Adjoint Method	0.485s	6.6612e-11	0.552s	**0.00008**
SBTO	15.647s	4.1923e-6	15.596s	0.00009
DeepONet MPC	144.306s	1.3723e-10	143.960s	**0.00008**
PDE-OP (Ours)	**0.223s**	1.5210e-6	**0.132s**	0.00009

Results. *Standard Targets (Table 10).* When controlling the temperature profile to a constant target, LMPC achieves machine precision (2.33×10^{-14}), whereas PDE-OP produces an MSE of 1.52×10^{-6}. However, for the ramp target, PDE-OP achieves accuracy (9.0×10^{-5}) on the same order of magnitude as the numerical optimizers (8.0×10^{-5}) while running significantly faster. Specifically, the inference time of PDE-OP is only $\mathbf{0.132}$ seconds for the ramp profile, compared to a runtime of ≈ 0.5 seconds for LMPC and Adjoint, and > 140 seconds for DeepONet MPC.

Advanced Targets (Table 11). For complex profiles, the performance gap narrows in accuracy but widens in efficiency. On the Step and Complex Gaussian targets, PDE-OP achieves an MSE of 1.36×10^{-2} and 2.42×10^{-2} respectively, effectively matching the optimal solutions found by LMPC and Adjoint methods. *Notably, PDE-OP computes these controls in* $\mathbf{0.03}$ *seconds, representing a* $\mathbf{10}\times$ *speedup over LMPC/Adjoint, a* $\mathbf{450}\times$ *speedup over SBTO, and a* $\mathbf{4000}\times$ *speedup over DeepONet MPC.*

Takeaway. For this experiment, we observe that classical control-based methods achieve the highest precision when targeting simple target profiles. However, PDE-OP offers real-time inference with negligible accuracy loss on complex, spatially heterogeneous targets, demonstrating its suitability for time-critical applications.

7.10.7 Optimal Control of 1D Heat Equation(Battery Thermal Management)

For this task, we evaluate the performance of each method on two categories of target profiles $y_{\text{target}}(x)$. **Standard Targets:** (i) *Sine* ($y_{\text{target}} \propto 0.6+0.3\sin(2\pi x)$); (ii) *Ramp* ($y_{\text{target}} = x+0.5$); and (iii) *Constant* ($y_{\text{target}} = 1$). **Advanced Targets:** (i) *Step* (discontinuous profile); (ii) *High-Frequency* (superposition of low and high frequency sine waves); and (iii) *Complex Gaussian* (stochastic mixture of Gaussian modes). We compare the proposed PDE-OP model against classical baselines (Adjoint Method, LMPC) and neural-based baselines (SBTO, DeepONet MPC). Figures 12 and 13 show the target profile and the terminal state produced by each method. From Figure 12, we observe that each method produces terminal states that closely align with the desired terminal state. As the target profiles become more complex, Figure 13 shows that all methods tend to capture only the overall trend, while high-frequency components are more challenging to approximate accurately.

Table 11: 1D Heat Equation. For each method and advanced target profile, we report runtime (s) and MSE.

	Step		High Frequency		Complex Gaussian	
Method	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**
LMPC	0.315s	**0.01359**	0.332s	**0.01868**	0.564s	**0.02405**
Adjoint Method	0.423s	**0.01359**	0.416s	**0.01868**	0.619s	**0.02405**
SBTO	15.595s	**0.01359**	15.556s	0.01892	15.555s	0.02408
DeepONet MPC	144.149s	**0.01359**	144.198s	**0.01868**	144.245s	**0.02405**
PDE-OP (Ours)	**0.033s**	0.01364	**0.033s**	0.02725	**0.033s**	0.02415

Table 12: 1D Heat Equation (Battery Physics). For each method and standard target profile, we report runtime (s) and MSE.

	Constant		Sine		Ramp	
Method	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**
LMPC	0.442s	**2.33e-14**	0.413s	**0.00035**	0.458s	0.00008
Adjoint Method	0.318s	1.05e-10	0.683s	**0.00035**	0.499s	0.00008
SBTO	13.260s	0.00010	14.265s	0.00077	13.315s	0.00020
DeepONet MPC	124.322s	1.66e-09	124.263s	0.00039	124.301s	**0.00008**
PDE-OP (Ours)	**0.030s**	9.56e-06	**0.031s**	0.00035	**0.030s**	0.00009

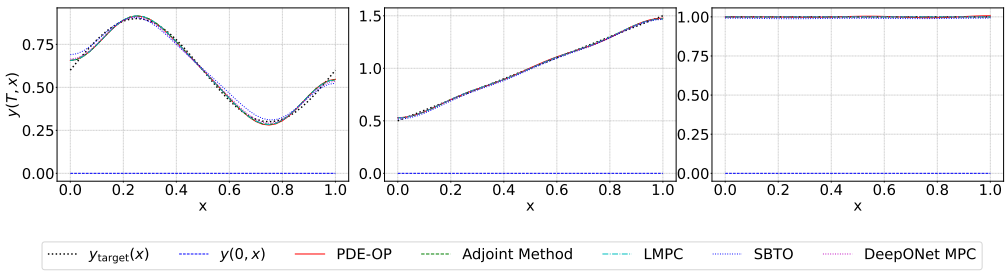

Figure 12: 1D Heat Equation (Battery Physics). Comparison of PDE-OP and other methods for three target profiles: $y_{\text{target}}(x) = 0.6 + 0.3(\sin(2\pi x))$(left), $y_{\text{target}}(x) = x + 0.5$(center), and $y_{\text{target}}(x) = 1$(right).

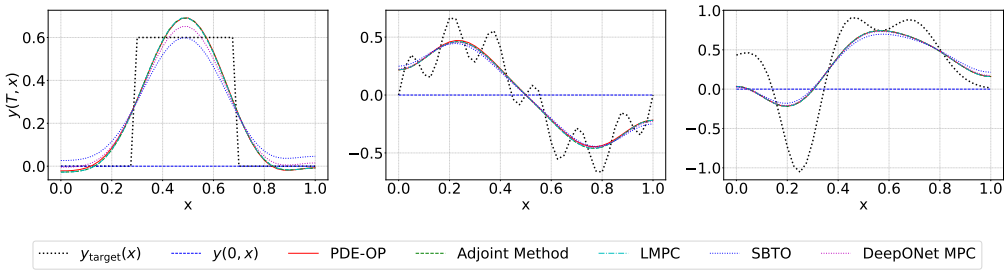

Figure 13: 1D Heat Equation (Battery Physics). Comparison of PDE-OP and baselines for advanced target profiles: *Step* (discontinuous)(left), *High-Frequency* ($y_{\text{target}}(x) = 0.5\sin(2\pi x) + 0.2\sin(12\pi x)$)(center), and *Complex Gaussian* (stochastic multi-modal profile)(right).

Results. We evaluate each method on the Lithium-Ion battery thermal management task with $D = 0.13$, which is $13\times$ higher than in the previous setting ($D = 0.01$), causing the heat to dissipate significantly faster and requiring more energy for control.

Standard Targets. Table 12 reports the MSE and computational time of each method given the Constant, Sine, and Ramp profiles. For the sinusoidal target, PDE-OP achieves an MSE of $3.52\times$

Table 13: 1D Heat Equation (Battery Physics). For each method and advanced target profile, we report runtime (s) and MSE.

	Step		High Frequency		Complex Gaussian	
Method	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**	**Runtime**	**Accuracy**
LMPC	0.359s	**0.01372**	0.397s	**0.02164**	0.533s	**0.11625**
Adjoint Method	0.404s	**0.01372**	0.505s	**0.02164**	0.438s	**0.11625**
SBTO	13.312s	0.01759	13.309s	0.02403	13.264s	0.12829
DeepONet MPC	124.198s	0.01462	124.180s	0.02238	124.260s	0.12037
PDE-OP (Ours)	**0.032s**	0.01382	**0.031s**	**0.02164**	**0.031s**	**0.11625**

10^{-4}, matching the accuracy of the Adjoint Method and LMPC (3.50×10^{-4}). For the Ramp target, PDE-OP yields an MSE of 9.78×10^{-5}, which is also comparable to the best baselines. In terms of speed, PDE-OP generates solutions in 0.030s, approximately $15\times$ faster than the Adjoint method (0.499s) and LMPC (0.458s).

Advanced Targets. Table 13 reports the MSE and computational time of each method in presence of complex target profiles. For the Step target, PDE-OP is close to the best baselines (0.01382 vs 0.01372). For the High-Frequency target, PDE-OP matches the best baselines, with an MSE of 0.02164. When the target is a *Complex Gaussian*, all methods incur a higher error (MSE $\approx 1.16 \times 10^{-1}$). This is due to the high diffusivity ($D = 0.13$), which prevents the formation of sharp, multi-modal peaks under the given constraints. PDE-OP matches the error of the Adjoint method (1.16×10^{-1}), but runs approximately $15\times$ faster than the best baselines.

Takeaway. On this physically realistic battery task, PDE-OP achieves the same solution quality as the optimal Adjoint and LMPC methods, even in presence of complex target profiles. Consistent with the other tasks, PDE-OP achieves speedups of one to two orders of magnitude over the most efficient numerical baselines, while running approximately 400× faster than SBTO and $4000\times$ faster than DeepONet MPC.

7.11 Discrete-Time Neural Operator with Direct Control Field $u(x,t)$

In this section we describe a variant of our discrete-time neural operator that learns *directly* the spatio–temporal control field $u(t,x)$ rather than the weights of the basis function as introduced in the *Discrete-time Neural Operator* section of the main text. Similarly to the original DeepONet Lu et al. (2021), PDE-OP's model $\mathcal{Y}_\theta$ factors into a *branch* network that encodes input functions and a *trunk* network that encodes query coordinates. At each time t_k, given the current state $y(t_k,x)$ and the applied control $u(t_k,x)$, this model predicts the next state at coordinate x:

$$\hat{\boldsymbol{y}}(t_{k+1},x) \;=\; \mathcal{Y}_\theta\big(t,\, x,\, \hat{\boldsymbol{y}}(t_k,x),\, \hat{\boldsymbol{u}}(t_k,x)\big) \tag{60}$$

Branch network. The branch net encodes the estimated system state $\hat{\boldsymbol{y}}(t_k,x)$ sampled at n fixed sensor locations together with estimated control signals $\hat{\boldsymbol{u}}(t_k,x)$, and maps this concatenated input to a d-dimensional vector of coefficients $\boldsymbol{b}(\hat{\boldsymbol{y}}(t_k,x),\hat{\boldsymbol{u}}(t_k)) \in \mathbb{R}^d$.
Trunk network. The trunk net encodes the spatial query coordinate x, producing a latent feature vector $\boldsymbol{\gamma}(x) \in \mathbb{R}^d$.
The estimated subsequent (in time) state $\hat{\boldsymbol{y}}(t_{k+1},x)$ at location x_i is obtained by combining the branch coefficients and trunk features through an inner product:

$$\hat{y}(t_{k+1},x_i) \;=\; \sum_{j=1}^{d} b_j(\hat{y}(t_k,x_i),\hat{\boldsymbol{u}}(t_k,x_i))\;\gamma_j(x_i), \tag{61}$$

for $i = 1,\dots,n$.

Model initialization and training (Direct u). In this setting the dynamic predictor $\mathcal{Y}_\theta$ takes as input the control action *sampled on the spatial grid*, together with the current state. $\mathcal{Y}_\theta$ is initialized in a supervised fashion by generating the target trajectories with a numerical PDE solver under diverse time-varying controls $u(t,x)$.

We sample admissible control sequences $\{u(t_k, x_i)\}$ on the grid $\{x_i\}_{i=1}^n$ from a zero-mean Gaussian random field (GRF) with prescribed spatial covariance (and optional temporal correlation), after which a clipping operation is applied to meet the actuator bounds $u_{\min} \le u(t_k, x_i) \le u_{\max}$. Rolling the solver with these inputs produces states $y(t_k, \cdot)$ satisfying (1b)-(1f). Each trajectory is converted into one-step training pairs

$$\big(y(t_k, x_i),\, u(t_k, x_i)\big) \;\longmapsto\; y(t_{k+1}, x_i),$$

for $k = 0, \dots, T-1$ and $i = 1, \dots, n$. Formally, the dataset consists of triples $\mathcal{D} = \Big\{\big(\boldsymbol{y}(t_k, x),\, \boldsymbol{u}(t_k, x),\, \boldsymbol{y}(t_{k+1}, x)\big)\Big\}$, with $\boldsymbol{y}(t_k, x) = [y(t_k, x_1), \dots, y(t_k, x_n)]^\top$ and $\boldsymbol{u}(t_k, x) = [u(t_k, x_1), \dots, u(t_k, x_n)]^\top$. $\mathcal{Y}_\theta$ is trained to minimize the following loss

$$\min_\theta \; \mathbb{E}_{(\boldsymbol{u}, y)\sim\mathcal{D}}\Big[\big\|\hat{y}(t, x) - y(t, x)\big\|^2\Big], \tag{62}$$

7.11.1 MPC FOR DIRECT CONTROL FIELD $u(x, t)$

Here we describe a "direct-u" variant for viscous Burgers with Dirichlet boundaries, where the control action is parameterized *at every spatial collocation point*. At each time t_k a vector, the optimization task consists of finding $\boldsymbol{u}(t_k, x) \in \mathbb{R}^{N_x}$ (e.g., for a given time instant t_k we have N_x spatial collocation point), and applying an implicit Crank–Nicolson step to propagate the state in time from t_k to t_{k+1}.

$$\begin{aligned} \min_{\{\boldsymbol{u}(t_i)\}_{i=k}^{k+N_p-1}} \quad & \sum_{i=k}^{k+N_p-1} \Big(\|\boldsymbol{y}(t_i) - \boldsymbol{y}_{\text{target}}\|^2_{\boldsymbol{T}_Q} + \|\boldsymbol{u}(t_i)\|^2_{\boldsymbol{R}}\Big) \\ & + \|\boldsymbol{y}(t_{k+N_p}) - \boldsymbol{y}_{\text{target}}\|^2_{\boldsymbol{Q}} \\ \text{s.t.} \quad & 42, \qquad u_{\min} \le \boldsymbol{u}(t_i) \le u_{\max}. \end{aligned} \tag{63}$$

In case of closed loop (NMPC), a horizon of length N_p yields $N_p\, N_x$ decision variables per iteration; we warm start across receding horizons and use bound-constrained quasi-Newton (e.g., L-BFGS-B/SLSQP). Here $\mathbf{T}_Q, \mathbf{R},$ and $\mathbf{Q}$ represent the weight matrices. To compute the gradients, we either backpropagate through the discrete CN integrator using automatic differentiation or use the fully discrete adjoint; both methods are exact but require one forward rollout and one backward sweep over the horizon. This grid-wise formulation removes any basis bias and shows the attainable tracking when the controller has full spatial freedom. However, its computational cost is higher than that of the basis-parameterized formulation: the per-step cost scales roughly with $N_p\, N_t\, N_x$, due to line-search/gradient evaluation entailing multiple CN solves. We therefore include it as a comprehensive baseline to contextualize the efficiency of our reduced-parameter (mode-based) controllers.

7.11.2 EXPERIMENT RESULTS FOR DIRECT CONTROL FIELD $u(t, x)$

When using the direct control field modeling for $u(t, x)$, we evaluate each method on the nonlinear, time-variant optimal control of the 1D Burgers' equation, using the same target profiles presented in the 1D Burgers' equation section of the main text. Tables 14-15 report the runtime (s) and terminal tracking error (MSE) of each method. All methods use the same spatial/temporal discretization and solver settings as in 10. Figure 14 shows the predicted velocity field (central sub-figure of Fig. 14) of PDE-OP's dynamic component $\mathcal{Y}_\theta$ at test time, which is compared to the solution (left sub-figure) obtained with a numerical PDE-solver. Figures 15 and 16 show the comparison of PDE-OP's predicted terminal state $\hat{y}(T, x)$ and those obtained using the baseline methods, over the same target profiles introduced in the main text. In figure 15, we set $N_p = 1$ for NMPC and $N_p = 10$ for the Adjoint-method, while in figure 16, we set $N_p = 4$ for NMPC and $N_p = 20$ for the Adjoint-method.

Results. Across all targets, PDE-OP is *two orders of magnitude faster* than classical methods, between 100–300$\times$ faster than the NMPC and the Adjoint. In terms of accuracy, for the *Sine* target, PDE-OP achieves the lowest MSE; for the *Parabola*, NMPC attains the best results with PDE-OP close behind; for the *Zero* target, the adjoint method achieves the lowest MSE with PDE-OP producing an MSE of the order of 10^{-5}.

Table 14: 1D Burgers' Equation with direct control field. PDE-OP vs. classical baselines with short horizon N_p. Here we use N_p=1 for NMPC and N_p=10 for adjoint. We report runtime (s) and MSE (lower is better).

	Parabola		**Sine**		**Constant**	
Method	Runtime	MSE	Runtime	MSE	Runtime	MSE
NMPC	39.254s	**0.00002**	76.368s	0.00014	36.483s	7.9141e−06
Adjoint-Method	47.805s	0.00124	79.644s	0.00317	24.956s	**2.7618e−08**
PDE-OP (ours)	**0.267s**	0.00003	**0.256s**	**0.00008**	**0.257s**	0.00008

Table 15: 1D Burgers' Equation with direct control field. PDE-OP vs. classical baselines with longer horizons. Here we use N_p=4 for NMPC and N_p=20 for adjoint. We report runtime (s) and MSE (lower is better).

	Parabola		**Sine**		**Constant**	
Method	Runtime	MSE	Runtime	MSE	Runtime	MSE
NMPC	826.176s	0.00016	3143.735s	0.00032	470.851s	0.00061
Adjoint-Method	186.910s	0.00129	263.891s	0.00316	51.702s	**4.9213e−09**
PDE-OP (ours)	**0.254s**	**0.00003**	**0.277s**	**0.00008**	**0.956s**	0.00008

Results. With N_p=4 (NMPC) and N_p=20 (adjoint), PDE-OP remains dramatically faster, three to four orders of magnitude—than classical solvers. For the *Parabola* target, PDE-OP is $\sim 3250\times$ faster than NMPC and $\sim 740\times$ faster than adjoint, while also reporting the best MSE (3×10^{-5}). For the *Sine* target, the speedups over classical methods is $\sim 11300\times$ vs. NMPC and $\sim 950\times$ vs. adjoint, while again attaining the lowest error (8×10^{-5}). For the *Zero* target, the Adjoint method attains near-perfect regulation (4.9×10^{-9}), with PDE-OP is still fastest, running in $0.956\,s$ and approximately $54\times$ faster than the Adjoint method and $495\times$ faster than NMPC, while reporting a very small tracking error (8×10^{-5}).

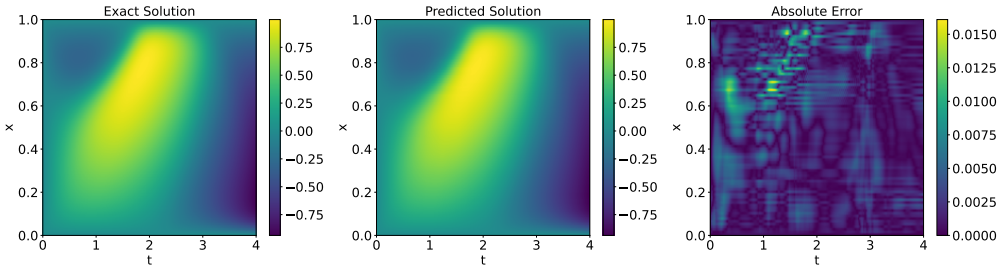

Figure 14: 1D Burgers' Equation with direct control field. PDE-OP's dynamic predictor $\mathcal{Y}_\theta$ solution estimate at test time: numerical solution (left), model prediction (center), and absolute error (right).

7.11.3 Ablation Study: Effect of the Number of Basis Functions M on PDE-OP's Performance

The parameter M (number of spatial basis functions) governs the spatial resolution and expressivity of the control input $u(t,x) = \sum_{i=1}^{M} c_i(t)\phi_i(x)$. A small M acts as a low-pass filter, restricting the controller to generate only smooth and low-frequency control signals. Conversely, a large M allows for fine-grained spatial control, which is theoretically necessary to approximate high-frequency or discontinuous target profiles (e.g., Step or Complex Gaussian). To quantify the trade-off between control expressivity and computational complexity, we train the PDE-OP controller with varying number of basis functions $M \in \{2, 4, 6, 8, 10\}$ to solve the optimal control problem for the 1D Heat Equation.

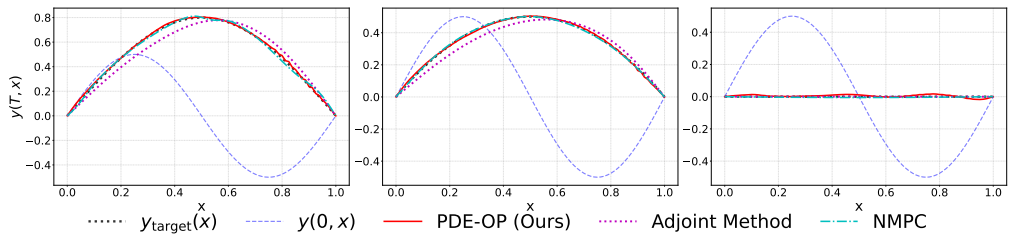

Figure 15: 1D Burgers' Equation with direct control field. Comparison between PDE-OP and each baseline method's solution on three different target profile: $y_{\text{target}}(x) = 0.8(\sin(\pi x))$ (left), $y_{\text{target}}(x) = 2x(1-x)$ (center), and $y_{\text{target}}(x) = 0$ (right). We set $N_p = 1$ for NMPC, and $N_p = 10$ for adjoint-method.

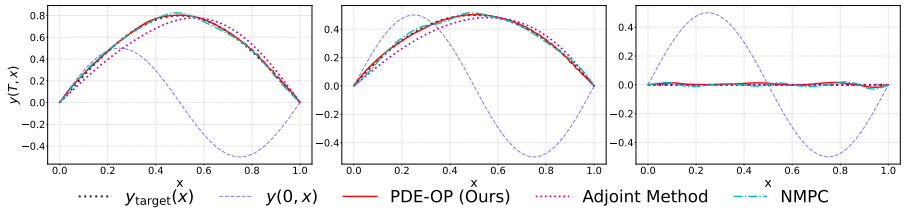

Figure 16: 1D Burgers' Equation with direct control field. Comparison between PDE-OP and each baseline method's solution on three different target profile: $y_{\text{target}}(x) = 0.8(\sin(\pi x))$ (left), $y_{\text{target}}(x) = 2x(1-x)$ (center), and $y_{\text{target}}(x) = 0$ (right). We set $N_p = 4$ for NMPC, and $N_p = 20$ for adjoint-method.

Results. As shown in Table 16 and Figure 17, increasing M results in a monotonic and significant reduction in the tracking error across all targets. For the smooth *Sine* target, the MSE drops from $0.014 (M = 2)$ to $0.001 (M = 10)$. The effect is most pronounced for the *Complex Gaussian* target, where the MSE drops from 0.268 $(M = 2)$ to 0.001 $(M = 10)$. This suggests that higher-dimensional bases are essential for capturing multi-modal shapes that contain significant high-frequency energy.

Crucially, increasing M incurs **negligible computational overhead**. The inference time remains stable at $\approx$ 0.029–0.031s regardless of M. This is because the neural operator processes inputs in parallel on the GPU, and the slight increase in the dimension of the basis projection matrix has minimal impact on the forward-pass latency. Thus, unlike classical optimization where cost scales with decision variables, PDE-OP allows for high-resolution control ($M = 10$) in real-time.

7.12 Ablation Study on Runtime Evaluation

To evaluate the trade-off between computational runtime and accuracy, we implemented an adaptive benchmarking to determine the minimum computational time required by each baseline method to achieve a solution quality comparable to those of our proposed method, PDE-OP.

To do so, we first run PDE-OP to set a baseline error threshold E_{thresh}. Let $\hat{u}_{\text{PDE-OP}}(t, x)$ be the control action generated by the surrogate controller $\mathcal{U}_\omega$, and let $y_{\text{PDE-OP}}(T, x)$ be the resulting terminal state computed using the discrete-time neural operator $\mathcal{Y}_\theta$. This error threshold is defined as the terminal tracking error (MSE):

$$E_{\text{thresh}} = \frac{1}{N_x} \left\| y_{\text{PDE-OP}}(T, x) - y_{\text{target}}(x) \right\|_2^2, \tag{64}$$

where $y_{\text{target}}(x)$ represents the prescribed target profile (as defined in the Experimental Setting section of the main paper). We then constrain each baseline to achieve a final error $\mathcal{L}_{\text{terminal}} \leq E_{\text{thresh}}$. In what follows, we describe the stopping criteria adopted by each method to match PDE-OP's error threshold E_{thresh} defined by (64).

Table 16: Ablation study on basis dimension M. We report the Final MSE (and inference runtime in parentheses) across three distinct target profiles. Increasing M significantly reduces error for complex targets without increasing computational cost.

Basis Dim (M)	Sine Accuracy (Runtime)	Step Accuracy (Runtime)	Complex Gaussian Accuracy (Runtime)
2	0.014 (0.029s)	0.084 (0.029s)	0.268 (0.030s)
4	0.001 (0.029s)	0.022 (0.029s)	0.117 (0.031s)
6	0.001 (0.029s)	0.013 (0.029s)	0.024 (0.031s)
8	0.001 (0.029s)	0.012 (0.029s)	0.006 (0.030s)
10	**0.001** (0.029s)	**0.007** (0.029s)	**0.001** (0.030s)

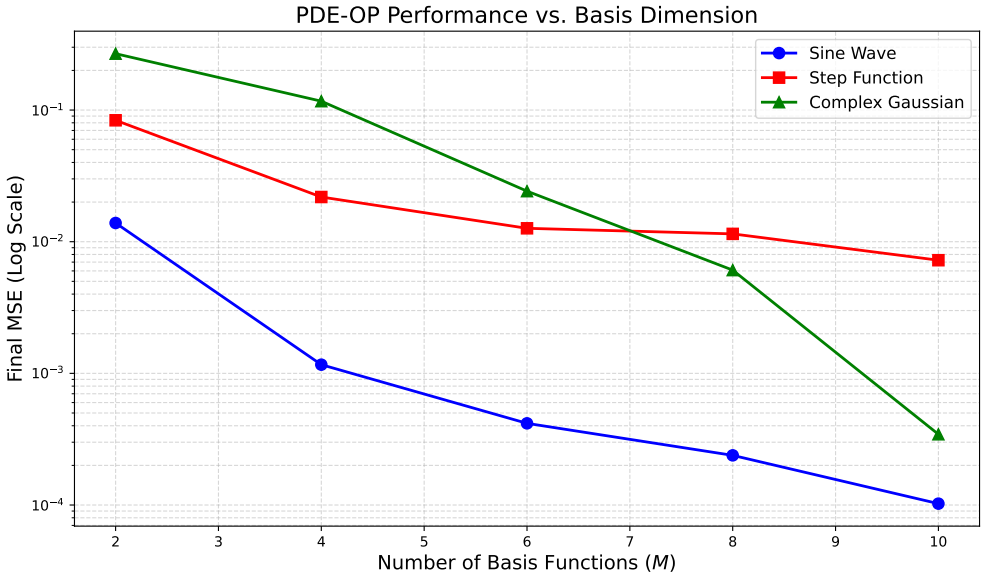

Figure 17: PDE-OP Performance vs. Basis Dimension(M)

1. Iterative Learning Baselines (Adaptive Early Stopping) For methods relying on iterative gradient-based optimization—specifically **SBTO** and **DeepONet MPC**—we implemented an online convergence check, detailed as follows.

- **SBTO:** The global optimization loop (optimizing the control weights $\mathbf{c}$) is terminated at iteration k if, given the predicted state $y_{\text{SBTO}}(T, x)$, the terminal loss satisfies the following:

$$\|y_{\text{SBTO}}(T, x) - y_{\text{target}}(x)\|_2^2 \leq E_{\text{thresh}} \tag{65}$$

- **Neural MPC:** The inner optimization loop at each time step t_k (planning the local horizon) is terminated early if the predicted trajectory error falls below E_{thresh}.

2. Classical Baselines (Tolerance Relaxation) To reduce the runtime of time-stepping methods like the **Adjoint Method** and **Linear/Nonlinear MPC**, we tuned their solver tolerances, since they require terminal target access and must integrate all the way to the final time step. We treated the solver tolerance ϵ (e.g., `ftol` for L-BFGS-B, `eps` for OSQP) as a hyperparameter and performed a logarithmic grid search to find the fastest setting satisfying the condition:

$$\begin{aligned} \min_{\epsilon} \quad & \text{Runtime}(\epsilon) \\ \text{s.t.} \quad & \|y_\epsilon(T, x) - y_{\text{target}}(x)\|_2^2 \leq E_{\text{thresh}}. \end{aligned} \tag{66}$$

Next, we report the runtime of the baseline methods run with the stopping criteria described above.

Results: Speed-Accuracy Trade-off Analysis. We conducted the ablation study using the *constant profile* setting, as it resulted in the most significant accuracy gap between PDE-OP and the

Table 17: **Runtime Evaluation (Constant Target).** Comparison of runtime (in seconds) required to match the target accuracy threshold (E_{thresh}) set by PDE-OP. Baseline methods run using the respective stopping criteria. In parenthesis we report the runtime time of each method running without the corresponding stopping criteria.

Method	**Voltage Eq.** *(Time in s)*	**1D Heat Eq.** *(Time in s)*	**1D Burgers' Eq.** *(Time in s)*
PDE-OP (Ours)	**0.037**	0.223	**0.176**
Direct / Linear / Nonlinear MPC†	1.567(2.092)	0.301(0.443)	59.463(132.243)
Adjoint Method	0.046(0.049)	**0.194**(0.485)	12.904(18.131)
SBTO (Early Stop)	0.123(0.508)	N/A(15.647)	2.461(66.074)
DeepONet MPC (Early Stop)	N/A	88.327(144.306)	46.342(98.296)

† *Direct Method used for Voltage; Linear MPC for Heat; Nonlinear MPC for Burgers.*

baseline methods. Table 17 reports the runtime required for each method to match the solution quality (E_{thresh}) of PDE-OP on the constant target profiles (i.e., $y_{\text{target}}(x) = 1$ for Voltage and 1D Heat equation, and $y_{\text{target}}(x) = 0$ for 1D Burgers' equation). When adopting the stopping criteria described above, we observe a modest reduction in runtime for each method. Despite so, PDE-OP continues to demonstrate a significant computational advantage, particularly in the nonlinear Burgers setting, which is also the most challenging benchmark considered here. For the **1D Burgers' Equation**, where nonlinearities pose a major challenge for classical solvers, the speedup is dramatic: PDE-OP (0.176s) is approximately 73× **faster than the Adjoint method** and 337× **faster than Nonlinear MPC**, even when those baselines are tuned to the loosest valid tolerance.

Notably, some entries in the table are marked as N/A, indicating that the corresponding method failed to pass the error threshold (E_{thresh}) achieved by PDE-OP (see Tables 5, 7, and 10).

8 Constraint-Satisfaction Metrics

This section defines the quantities reported in Table 2 of the main text and documents their validation. Each metric evaluates one block of $\boldsymbol{h}'$ in Eq. (9) or the box constraints of Eq. (1f). All residuals are made dimensionless by reporting

$$\mathrm{rel}(\boldsymbol{r}, \boldsymbol{d}) = \frac{\|\boldsymbol{r}\|_2}{\|\boldsymbol{d}\|_2 + \epsilon}, \qquad \epsilon = 10^{-12}. \tag{67}$$

8.1 Dynamics residual

The residual of the predicted trajectory with respect to the same discrete operator used to generate the training data. This way, any discrepancy reflects inconsistency with the same discrete operator used to generate the data.

For the 2D heat equation the exact discrete dynamics are $\boldsymbol{y}_{k+1} = A\boldsymbol{y}_k + B\boldsymbol{c}_k + \boldsymbol{g}$, with $A, B, \boldsymbol{g}$ from the Crank–Nicolson discretization. The residual is

$$\boldsymbol{r}_k = \hat{\boldsymbol{y}}_{k+1} - (A\hat{\boldsymbol{y}}_k + B\hat{\boldsymbol{c}}_k + \boldsymbol{g}), \tag{68}$$

reported as $\mathrm{rel}(\boldsymbol{r}_k, A\hat{\boldsymbol{y}}_k + B\hat{\boldsymbol{c}}_k + \boldsymbol{g})$.

For the Burgers' equation, with A_{cn}, B_{cn} the Crank–Nicolson matrices carrying the Dirichlet rows and $\mathcal{A}(\cdot)$ the upwind advection operator,

$$\boldsymbol{r}_k = A_{cn}\hat{\boldsymbol{y}}_{k+1} - (B_{cn}\hat{\boldsymbol{y}}_k - \Delta t\, \mathcal{A}(\hat{\boldsymbol{y}}_k) + \Delta t\, \boldsymbol{f}_k), \tag{69}$$

where $\boldsymbol{f}_k = \hat{\boldsymbol{c}}_k^\top \boldsymbol{\phi}$ is the basis-expanded forcing.

For the voltage equation, with A, B the Crank–Nicolson operators carrying the Neumann ghost-node rows,

$$\boldsymbol{r}_k = A\hat{\boldsymbol{y}}_{k+1} - (B\hat{\boldsymbol{y}}_k + \Delta t\, (\alpha \boldsymbol{u} + \beta y_{\text{ref}})). \tag{70}$$

Table 18: Metric values computed on ground-truth solver trajectories. Each reference value is evaluated using the same metric and on the same grid as the corresponding model prediction, ensuring a direct comparison. The last two rows instead compute the dynamics residual on the solver's native grid, confirming that the residual stencils exactly reproduce the solver update. Thus, the nonzero reference values reported above arise from evaluating the solver trajectory on the prediction grid rather than from numerical solver tolerances. The Neumann boundary metrics are nonzero because the solver enforces Neumann conditions through ghost nodes, yielding truncation-order discretization errors even for the exact discrete solution.

System	Quantity	GT value
Voltage	dynamics residual	1.4×10^{-2}
2D Heat	dynamics residual	0 (exact)
Burgers	dynamics residual	8.8×10^{-4}
Voltage	Neumann BC residual	2.4×10^{-2}
2D Heat	Neumann BC residual	3.1×10^{-2}
Burgers	Dirichlet BC residual	7.4×10^{-18}
All	initial condition	0
All	bound violation	0
Voltage	dyn. residual, solver grid	4.2×10^{-15}
Burgers	dyn. residual, solver grid	4.5×10^{-16}

8.2 Initial and boundary conditions

The initial-condition error is $\mathrm{RMS}(\hat{\boldsymbol{y}}_0 - \boldsymbol{y}_0)$. For Dirichlet boundaries,

$$\mathrm{BC}_{\mathrm{RMS}} = \sqrt{\frac{1}{2N_t} \sum_k \left[(\hat{y}_k^{(0)} - v)^2 + (\hat{y}_k^{(-1)} - v)^2 \right]}, \quad v = 0. \tag{71}$$

For Neumann boundaries we use second-order one-sided differences,

$$\partial_x \hat{y}\big|_{x_0} \approx \frac{-3\hat{y}_0 + 4\hat{y}_1 - \hat{y}_2}{2\Delta x}, \;\; \partial_x \hat{y}\big|_{x_{-1}} \approx \frac{3\hat{y}_{-1} - 4\hat{y}_{-2} + \hat{y}_{-3}}{2\Delta x}, \tag{72}$$

and report the RMS over all boundary points and times.

Box-constraint violations. For each scalar control entry, we define the violation magnitude as

$$v_i = \max(0,\; u_i - u_{\max},\; u_{\min} - u_i).$$

We characterize constraint satisfaction using the violation rate

$$r_v = \frac{1}{n} \left| \{ i : v_i > 0 \} \right|$$

and the mean magnitude conditional on a violation,

$$\bar{v}_+ = \frac{1}{|\{ i : v_i > 0 \}|} \sum_{i : v_i > 0} v_i,$$

with $\bar{v}_+ = 0$ when no violation occurs. These quantities correspond to the *Violation Rate* and *Mean Control Violation* columns in Table 2.

8.3 Validation against the ground-truth solver

Every metric was first evaluated on trajectories produced by the numerical solver to establish a reference (GT) floor. Table 18 reports these values, which are also shown alongside each constraint metric in the main text.

9 Additional Results on Constraint Satisfaction

Table 2 in the main text summarizes replay accuracy, physics-constraint residuals, boundary-condition errors, and control-bound violations over 40 unseen instances. This section provides additional distributional information on the discrepancy between surrogate predictions and numerical-solver replay, together with a more detailed analysis of the control-bound violations.

Table 19: Surrogate-to-replay discrepancy over the same 40 unseen instances used in the main experiments. The ratio column reports the median and IQR of $e_{\mathrm{true}}/e_{\mathrm{surr}}$. The normalized gap is $1 - e_{\mathrm{surr}}/e_{\mathrm{true}}$. The last column reports the number of instances for which replay in the numerical solver increases the terminal error.

System	Ratio Median [IQR]	Median Normalized Gap (%)	Instances ($e_{\mathrm{true}} > e_{\mathrm{surr}}$)
Voltage	25.74 $[11.73, 65.83]$	96.1%	$39/40$
2D Heat	2.74 $[1.16, 33.26]$	63.5%	$40/40$
Burgers	1.01 $[0.997, 1.45]$	0.99%	$27/40$

Surrogate-to-replay discrepancy. For each unseen instance, we first compute the ratio

$$r_i = \frac{e_{\mathrm{true},i}}{e_{\mathrm{surr},i}},$$

where $e_{\mathrm{surr},i}$ is the terminal MSE predicted by the surrogate and $e_{\mathrm{true},i}$ is the terminal MSE obtained by replaying the same predicted control in the numerical PDE solver. Because $e_{\mathrm{surr},i}$ can be close to zero, the raw ratio can become large. We therefore report it together with the replay-normalized surrogate gap

$$g_i = 1 - \frac{e_{\mathrm{surr},i}}{e_{\mathrm{true},i}} = \frac{e_{\mathrm{true},i} - e_{\mathrm{surr},i}}{e_{\mathrm{true},i}}.$$

Here, $g_i = 0$ indicates exact agreement, $g_i > 0$ indicates that the surrogate underestimates the replay error, and $g_i < 0$ indicates that it overestimates the replay error.

Table 19 shows that Burgers has the smallest discrepancy, with a median ratio of 1.01 and a median normalized gap of only 0.99%. The discrepancy increases for 2D Heat, whose median ratio is 2.74 and normalized gap is 63.5%. Voltage exhibits the strongest surrogate optimism, with a median ratio of 25.74 and a normalized gap of 96.1%. The corresponding median absolute increases in terminal MSE, $e_{\mathrm{true}} - e_{\mathrm{surr}}$, are 8.22×10^{-4} for Voltage, 2.84×10^{-2} for 2D Heat, and 5.47×10^{-5} for Burgers. These results are consistent with surrogate fidelity being a major contributor to replay degradation, although the effect cannot be fully separated from the controls produced by the learned optimizer.

Bound violations. Violations are sparse at the level of individual control entries, but their magnitude and distribution across instances differ substantially among the three systems. For Voltage, 35 of 4000 control entries violate the admissible interval, corresponding to a violation rate of 0.875%. Violations occur in 5 of the 40 instances, with a conditional mean magnitude of 0.0226 and a maximum excursion of 0.0681.

For 2D Heat, 356 of 26240 control entries violate the bounds, giving a violation rate of 1.36%. Although the entry-wise rate remains small, violations occur in 39 of the 40 instances. Their conditional mean magnitude is 0.220, and the largest observed excursion is 2.034.

Burgers exhibits the smallest violation magnitudes: 234 of 32000 control entries violate the bounds, corresponding to a rate of 0.731%. These violations occur in 8 of the 40 instances, with a conditional mean magnitude of 0.00807 and a maximum excursion of 0.0671.

10 Hyperparameter Settings

Tables 20 - 23 report values of the hyperparameters of PDE-OP and the baseline methods related to the voltage optimization experiments; Tables 24 - 28 report the values of the hyperparameters of PDE-OP and the baselines related to the 1D heat equation experiments;Tables 29 - 33 report the values of the hyperparameters of PDE-OP and the baselines related to the 2D heat equation experiments; Tables 34 - 38 report the values of the hyperparameters of PDE-OP and baseline methods for the 1D Burgers' equation experiments; Tables 39 - 42 report the values of the hyperparameters of PDE-OP and the baselines related to the 1D Burgers' with varying coefficient equation experiments. Each method adopts the same PDE, grid, and horizon parameters $n, N_x, N_t, [u_{\min}, u_{\max}], [c_{\min}, c_{\max}]$.

Table 20: Voltage Equation: simulation parameters and model/training hyperparameters. *MLP* Goodfellow et al. (2016) a fully connected feedforward network.

Category	Hyperparameter	Value
PDE & Simulation	Domain length L	1.0
	Final time T	5.0
	Diffusion D	0.1
	Leakage β	1.0
	Control gain α	2.0
	Initial state $V(x, 0)$	0.0
	Reference $V_{\text{ref}}(x)$	1.0
	Control bounds $[u_{\min}, u_{\max}]$	$[-1.0,\ 1.0]$
Discretization	Solver grid (n)	101
	Solver steps (N_t)	101
	Number of sensors $(N_x = n)$	101
PDE-OP's $\mathcal{Y}_\theta$	Number of Train Trajectories	1000
	Number of Test/Val Trajectories	100
	Learning Rate	5×10^{-4}
	Number of epochs	1000
	Optimizer	Adam
	Batch size	1024
	Activation function	SiLU
	Lagrange Multiplier ρ	0.05
	Branch net (b): # layers	4
	Branch net (b): hidden sizes	$[512, 512, 512, 512]$
	Trunk net (γ): # layers	4
	Trunk net (γ): hidden sizes	$[512, 512, 512, 512]$
	Feature dimension (d)	512
PDE-OP's $\mathcal{U}_\omega$	Learning rate	1×10^{-3}
	Number of epochs	1000
	Optimizer	AdamW
	Batch size	256
	Activation function	RELU
	MLP: # layers	3
	MLP: hidden sizes	$[256, 256, 256]$

Table 21: Voltage Equation: Direct(finite-difference) method parameters.

Category	Hyperparameter	Value / Note
Optimizer	Algorithm	L-BFGS-B (box bounds $[u_{\min}, u_{\max}]$)
	Max iterations	100
	Tolerances	`ftol` $= 10^{-6}$ (and/or `gtol`)
	Initialization	$\boldsymbol{u}(t_0) = \mathbf{0}$ (uniform)
Finite diff.	Stencil	forward (optionally central)
	Step size	$\epsilon_j = \eta(1 + \lvert u_j \rvert)$, $\eta = 10^{-6}$
Cost eval	Rollout solver	CN
	Runtime note	$N_x{+}1$ evals/grad (or $2N_x$ for central)

Table 22: Voltage Equation: Adjoint-method parameters.

Category	**Hyperparameter**	**Value / Note**
Optimizer	Algorithm	L-BFGS-B (box bounds $[u_{\min}, u_{\max}]$)
	Max iterations	100
	Tolerances	$\texttt{ftol} = 10^{-6}$ (and/or $\texttt{gtol}$)
	Initialization	$\boldsymbol{u}(t_0) = \mathbf{0}$
Adjoint solves	Forward stepper	CN
	Backward sweep	discrete adjoint (one pass)
Numerics	Linear solves	direct (LU) or iterative tol 10^{-10}

Table 23: Voltage Equation: SBTO (Surrogate-Based) method parameters.

Category	**Hyperparameter**	**Value / Note**
Optimizer	Algorithm	Adam
	Learning Rate	1×10^{-2}
	Max iterations	200
	Initialization	Latent parameters $\mathbf{0}$ (maps to $\boldsymbol{u} = \mathbf{0}$)
Constraints	Method	Soft saturation ($\tanh(\cdot) \times u_{\max}$)
Loss	Formulation	MSE (tracking) + L_2 penalty
	Effort weight	1×10^{-5}
Surrogate	Model	DeepONet (Frozen parameters)
	Gradient	Automatic Differentiation

Table 24: 1D Heat Equation: simulation parameters and model/training hyperparameters. *LSTM* Hochreiter & Schmidhuber (1997) denotes a Long Short-Term Memory network.

Category	**Hyperparameter**	**Value**
PDE & Simulation	Domain length L	1.0
	Final time T	1.0
	Diffusion D	0.01
	Leakage β	0.5
	Control gain α	2.0
	Initial state $V(x, 0)$	0.0
	Reference $V_{\text{ref}}(x)$	0.0
	Weight bounds $[c_{\min}, c_{\max}]$	$[-1.0,\ 1.0]$
Discretization	Solver grid (n)	41
	Solver steps (N_t)	41
	Number of sensors $(N_x = n)$	41
	Number of basis functions (M)	6
PDE-OP's $\mathcal{Y}_\theta$	Learning Rate	1×10^{-3}
	Number of Train Trajectories	5000
	Number of Test/Val Trajectories	1000
	Number of epochs	2000
	Optimizer	AdamW
	Batch size	128
	Activation function	RELU
	Branch net (b): # layers	4
	Branch net (b): hidden sizes	$[512, 512, 512, 512]$
	Trunk net (γ): # layers	4
	Trunk net (γ): hidden sizes	$[512, 512, 512, 512]$
	Feature dimension (d)	512
PDE-OP's $\mathcal{U}_\omega$	Learning rate	1×10^{-4}
	Number of epochs	5000
	Optimizer	AdamW
	Batch size	2048
	Activation function	RELU
	LSTM: # layers	2
	LSTM: hidden sizes	$[256, 256]$

Table 25: 1D Heat Equation: Linear MPC (LMPC) parameters. $(^*)$ The basis function are used by each method.

Category	Parameter	Value / Note
Horizon	Prediction length N_p	10
Dynamics	State update	$\boldsymbol{y}(t_{k+1}) = \boldsymbol{A}\boldsymbol{y}(t_k) + \boldsymbol{B}\boldsymbol{u}(t_k) + \boldsymbol{g}$
	Discretization	Crank–Nicolson (linear)
Actuation	Basis*	$\cos(j\pi x/L)$, $j = 0, \ldots, m-1$
Solver	QP solver	OSQP (CVXPY), warm start
Stopping	Tolerances	default OSQP (primal/dual res.)

Table 26: 1D Heat Equation: Adjoint method parameters.

Category	Parameter	Value / Note
Horizon	Prediction length N_p	10
Forward	Integrator	Crank–Nicolson (linear), same $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{g}$
Adjoint	Backward recursion	$\boldsymbol{\lambda}_{N_t} = \boldsymbol{Q}\left(\boldsymbol{y}_{N_t} - \boldsymbol{y}_{\text{target}}\right)$
		$\boldsymbol{\lambda}_k = \boldsymbol{A}^\top \boldsymbol{\lambda}_{k+1} + \boldsymbol{T}_Q\left(\boldsymbol{y}(t_k) - \boldsymbol{y}_{\text{target}}\right)$
Gradient	Per step	$\nabla_{\boldsymbol{u}(t_k)} J_h = \boldsymbol{R}\,\boldsymbol{u}(t_k) + \boldsymbol{B}^\top \boldsymbol{\lambda}_{k+1}$.
Optimizer	Algorithm	L-BFGS-B, warm start
Stopping	Max iters / tolerances	100; $\texttt{ftol} = 10^{-6}$

Table 27: 1D Heat Equation: SBTO (Surrogate-Based) method parameters.

Category	Hyperparameter	Value / Note
Optimizer	Algorithm	Adam
	Learning Rate	1×10^{-2}
	Max iterations	200
Loss	Formulation	Terminal MSE + Running MSE + Control Effort
	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Constraints	Method	Soft saturation $(\tanh(\cdot))$
Surrogate	Model	Propagator DeepONet (Frozen)
	Rollout	Recurrent (Full Horizon N_t)

Table 28: 1D Heat Equation: DeepONet MPC parameters.

Category	Hyperparameter	Value / Note
Horizon	Prediction length N_p	10
Optimizer	Algorithm	Adam (per time step)
	Learning Rate	2×10^{-2}
	Steps per solve	200
Loss	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Control	Strategy	Receding Horizon (Apply 1st action)
	Parameterization	Basis functions (optimized via backprop)
Surrogate	Model	Propagator DeepONet (Frozen)

Table 29: 2D Heat Equation: simulation parameters and model/training hyperparameters. *LSTM* Hochreiter & Schmidhuber (1997) denotes a Long Short-Term Memory network.

Category	Hyperparameter	Value
PDE & Simulation	Domain sizes L_x, L_y	$1.0, 1.0$
	Final time T	1.0
	Diffusion D	0.01
	Leakage β	0.5
	Control gain α	2.0
	Initial state $V(x, y, 0)$	0.0
	Reference V_{ref}	0.0
	Weight bounds $[c_{\min}, c_{\max}]$	$[-1.0,\ 1.0]$
Discretization	Solver grid $(N_x \times N_y)$	21×21
	Solver steps (N_t)	41
	Number of sensors (N_{sens})	441 (21×21)
	Number of basis functions (M)	16 (4×4)
PDE-OP's $\mathcal{Y}_\theta$	Learning Rate	1×10^{-3}
	Number of Train Trajectories	5000
	Number of Test/Val Trajectories	1000
	Number of epochs	2000
	Optimizer	AdamW
	Batch size	64
	Lagrange Multiplier ρ	0.05
	Activation function	RELU
	Branch net (b): # layers	4
	Branch net (b): hidden sizes	$[512, 512, 512, 512]$
	Trunk net (γ): # layers	2
	Trunk net (γ): hidden sizes	$[128, 128]$
	Feature dimension (d)	128
PDE-OP's $\mathcal{U}_\omega$	Learning rate	1×10^{-5}
	Number of epochs	5000
	Optimizer	AdamW
	Batch size	1024
	Activation function	RELU
	LSTM: # layers	2
	LSTM: hidden sizes	$[256, 256]$

Table 30: 2D Heat Equation: Adjoint method parameters.

Category	Parameter	Value / Note
Horizon	Prediction length N_p	10
Forward	Integrator	Crank–Nicolson (2D linear)
	Operator $\mathbf{A}$	Derived from $\mathbf{I} - \frac{D\Delta t}{2}\mathbf{L}_{2D}$
Adjoint	Backward recursion	$\boldsymbol{\lambda}_{N_t} = \boldsymbol{Q}\,(\boldsymbol{y}_{N_t} - \boldsymbol{y}_{\text{target}})$
		$\boldsymbol{\lambda}_k = \mathbf{A}^\top \boldsymbol{\lambda}_{k+1} + \boldsymbol{T}_Q\,(\boldsymbol{y}(t_k) - \boldsymbol{y}_{\text{target}})$
Gradient	Per step	$\nabla_{\boldsymbol{u}(t_k)} J_h = \boldsymbol{R}\,\boldsymbol{u}(t_k) + \mathbf{B}^\top \boldsymbol{\lambda}_{k+1}$.
Optimizer	Algorithm	L-BFGS-B, warm start
Stopping	Max iters / tolerances	30; `ftol` $= 10^{-6}$

Table 31: 2D Heat Equation: Linear MPC (LMPC) parameters.

Category	Parameter	Value / Note
Horizon	Prediction length N_p	10
Dynamics	State update	$\boldsymbol{y}(t_{k+1}) = \mathbf{A}\boldsymbol{y}(t_k) + \mathbf{B}\boldsymbol{u}(t_k) + \boldsymbol{g}$
	Discretization	Crank–Nicolson (2D Kronecker)
Actuation	2D Basis*	$\cos(\frac{i\pi x}{L_x})\cos(\frac{j\pi y}{L_y})$
Solver	QP solver	OSQP (CVXPY), warm start
Stopping	Tolerances	default OSQP (primal/dual res.)

Table 32: 2D Heat Equation: SBTO (Surrogate-Based) method parameters.

Category	Hyperparameter	Value / Note
Optimizer	Algorithm	Adam
	Learning Rate	1×10^{-2}
	Max iterations	200
Loss	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Constraints	Method	Soft saturation ($\tanh(\cdot)$)
Surrogate	Model	Propagator DeepONet (Frozen)
	Inputs	2D Coordinates (x, y) + Basis Weights

Table 33: 2D Heat Equation: DeepONet MPC parameters.

Category	Hyperparameter	Value / Note
Horizon	Prediction length N_p	10
Optimizer	Algorithm	Adam (per step)
	Learning Rate	2×10^{-2}
	Steps per solve	25
Loss	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Control	Strategy	Receding Horizon (Apply 1st action)
Surrogate	Model	Propagator DeepONet (Frozen)

Table 34: 1D Burgers' Equation: simulation parameters and model/training hyperparameters. *LSTM* Hochreiter & Schmidhuber (1997) denotes a Long Short-Term Memory network.

Category	Hyperparameter	Value
PDE & Simulation	Domain length L	1.0
	Final time T	4.0
	Viscosity ν	0.03
	Weight bounds $[c_{\min}, c_{\max}]$	$[-1.0,\ 1.0]$
Discretization	Solver grid (n)	81
	Solver steps (N_t)	201
	Number of sensors $(N_x = n)$	81
	Number of basis functions (M)	4
PDE-OP's $\mathcal{Y}_\theta$	Learning Rate	1×10^{-3}
	Number of Train Trajectories	5000
	Number of Test/Val Trajectories	1000
	Number of epochs	2000
	Optimizer	AdamW
	Batch size	512
	Lagrange Multiplier ρ	0.05
	Activation function	RELU
	Branch net (b): # layers	3
	Branch net (b): hidden sizes	$[256, 256, 256]$
	Trunk net (γ): # layers	5
	Trunk net (γ): hidden sizes	$[256, 256, 256, 256, 256]$
	Feature dimension (d)	128
PDE-OP's $\mathcal{U}_\omega$	Learning rate	1×10^{-4}
	Number of epochs	5000
	Optimizer	AdamW
	Batch size	2048
	Activation function	RELU
	LSTM: # layers	2
	LSTM: hidden sizes	$[256, 256]$

Table 35: 1D Burgers' Equation: Nonlinear MPC (NMPC) parameters. $(^*)$ The basis functions are used by each method.

Category	Parameter	Value / Note
Horizon	Prediction length N_p	e.g. 10 (receding horizon)
Dynamics	Implicit step (CN)	$\boldsymbol{y}(t_{k+1}) = \mathcal{G}(\boldsymbol{y}(t_k), \boldsymbol{c}(t_k))$
Actuation	Basis*	$\sin\big((j{+}1)\pi x/X\big),\ j = 0, \ldots, M-1$
Solver	Nonlinear optimizer	SLSQP (single-shooting), box bounds; warm start by shift
CN inner	Newton solve per step	tolerance 10^{-10}, max iters 25

Table 36: 1D Burgers' Equation: Adjoint method parameters.

Category	Parameter	Value / Note
Horizon	Prediction lentgh N_p	10
Forward	Implicit CN dynamics	as in NMPC, Dirichlet enforced each step
Jacobian	Semi-discrete residual	$J(\boldsymbol{y}) = -\boldsymbol{D}\,\mathrm{diag}(\boldsymbol{y}) + \nu\,\boldsymbol{D}^2$
Adjoint	Backward sweep	$A_k^\top \boldsymbol{q}_k = \Delta t\,(\boldsymbol{y}(t_{k+1}) - \boldsymbol{y}_{\text{target}})$; $\boldsymbol{p}_k = \Delta t\,(\boldsymbol{y}(t_k) - \boldsymbol{y}_{\text{target}}) - B_k^\top \boldsymbol{q}_k$
Gradient	Per-step control gradient	$\nabla_{\boldsymbol{u}(t_k)} J = \Delta t\big(\alpha\,\boldsymbol{u}(t_k) + \boldsymbol{B}^\top \boldsymbol{q}_k\big)$,
Optimizer	Outer optimizer	L-BFGS-B, warm start

Table 37: 1D Burgers' Equation: SBTO (Surrogate-Based) method parameters.

Category	Hyperparameter	Value / Note
Optimizer	Algorithm	Adam
	Learning Rate	1×10^{-2}
	Max iterations	150
Loss	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Constraints	Method	Soft saturation ($\tanh(\cdot) \times c_{\max}$)
Surrogate	Model	Propagator DeepONet (Frozen)
	Rollout	Recurrent (Full Horizon N_t)

Table 38: 1D Burgers' Equation: DeepONet MPC parameters.

Category	Hyperparameter	Value / Note
Horizon	Prediction length N_p	10
Optimizer	Algorithm	Adam (per time step)
	Learning Rate	2×10^{-2}
	Steps per solve	25
Loss	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Control	Strategy	Receding Horizon (Apply 1st action)
Surrogate	Model	Propagator DeepONet (Frozen)

Table 39: 1D Burgers' Equation with varying coefficient: simulation parameters and model/training hyperparameters. *LSTM* Hochreiter & Schmidhuber (1997) denotes a Long Short-Term Memory network.

Category	Hyperparameter	Value
PDE & Simulation	Domain length L	1.0
	Final time T	4.0
	Viscosity $\nu(x)$ range	$[0.01, 0.05]$
	Correlation length (GRF)	0.3
	Control scale	2.0
Discretization	Solver grid (N_x)	81
	Solver steps (N_t)	401
	Number of sensors (N_{sens})	81
	Number of basis functions (M)	4
PDE-OP's $\mathcal{Y}_\theta$	Learning Rate	1×10^{-3}
	Number of Train Trajectories	1000
	Number of Test/Val Trajectories	200
	Number of epochs	2000
	Optimizer	AdamW
	Batch size	512
	Activation function	RELU
	Branch net (b): # layers	3
	Branch net (b): hidden sizes	$[256, 256, 256]$
	Trunk net (γ): # layers	5
	Trunk net (γ): hidden sizes	$[256, 256, 256, 256, 256]$
	Feature dimension (d)	256
PDE-OP's $\mathcal{U}_\omega$	Learning rate	1×10^{-3}
	Number of epochs	5000
	Optimizer	Adam
	Batch size	1024
	Activation function	RELU
	LSTM: # layers	2
	LSTM: hidden sizes	$[256, 256]$

Table 40: 1D Burgers' Equation with varying coefficient: Adjoint method parameters.

Category	Parameter	Value / Note
Horizon	Prediction length N_p	10
Forward	Integrator	Implicit Crank–Nicolson (Newton solver)
	Operator $\mathbf{L}_\nu$	Spatially varying viscosity $\nu(x)$
Adjoint	Backward recursion	Linearized adjoint equation $\boldsymbol{p}_k = \Delta t(\boldsymbol{y}_k - \boldsymbol{y}_{\text{ref}}) - \mathbf{B}_k^\top \boldsymbol{q}_k$
Gradient	Per step	$\nabla_{\boldsymbol{u}} J = \Delta t(\alpha \boldsymbol{u}_k + \mathbf{B}^\top \boldsymbol{q}_k)$
Optimizer	Algorithm	L-BFGS-B, warm start
Stopping	Max iters / tolerances	40; `ftol` $= 10^{-6}$

Table 41: 1D Burgers' Equation with varying coefficient: SBTO (Surrogate-Based) method parameters.

Category	**Hyperparameter**	**Value / Note**
Optimizer	Algorithm	Adam
	Learning Rate	1×10^{-2}
	Max iterations	150
Loss	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Constraints	Method	Soft saturation ($\tanh(\cdot) \times 2.0$)
Surrogate	Model	Propagator DeepONet (Frozen)
	Inputs	Viscosity profile $\nu(x)$ included

Table 42: 1D Burgers' Equation with varying coefficient: DeepONet MPC parameters

Category	**Hyperparameter**	**Value / Note**
Horizon	Prediction length N_p	10
Optimizer	Algorithm	Adam (per time step)
	Learning Rate	2×10^{-2}
	Steps per solve	25
Loss	Weights	$\lambda_{term} = 1.0, \lambda_{run} = 0.1, \lambda_{effort} = 1 \times 10^{-5}$
Control	Strategy	Receding Horizon (Apply 1st action)
Surrogate	Model	Propagator DeepONet (Frozen)