DiffPhyCon: A Generative Approach to Control Complex Physical Systems

Long Wei[1*] **Peiyan Hu**[2*§] **Ruiqi Feng**[1*] **Haodong Feng**[1] **Yixuan Du**[3§] **Tao Zhang**[1]
Rui Wang[4§] **Yue Wang**[5] **Zhi-Ming Ma**[2] **Tailin Wu**[1†]
[1]School of Engineering, Westlake University,
[2]Academy of Mathematics and Systems Science, Chinese Academy of Sciences,
[3]Jilin University, [4]Fudan University, [5]Microsoft AI4Science
{weilong,hupeiyan,fengruiqi,wutailin}@westlake.edu.cn

Abstract

Controlling the evolution of complex physical systems is a fundamental task across science and engineering. Classical techniques suffer from limited applicability or huge computational costs. On the other hand, recent deep learning and reinforcement learning-based approaches often struggle to optimize long-term control sequences under the constraints of system dynamics. In this work, we introduce Diffusion Physical systems Control (DiffPhyCon), a new class of method to address the physical systems control problem. DiffPhyCon excels by simultaneously minimizing both the learned generative energy function and the predefined control objectives across the entire trajectory and control sequence. Thus, it can explore globally and plan near-optimal control sequences. Moreover, we enhance DiffPhyCon with prior reweighting, enabling the discovery of control sequences that significantly deviate from the training distribution. We test our method on three tasks: 1D Burgers' equation, 2D jellyfish movement control, and 2D high-dimensional smoke control, where our generated jellyfish dataset is released as a benchmark for complex physical system control research. Our method outperforms widely applied classical approaches and state-of-the-art deep learning and reinforcement learning methods. Notably, DiffPhyCon unveils an intriguing fast-close-slow-open pattern observed in the jellyfish, aligning with established findings in the field of fluid dynamics. The project website, jellyfish dataset, and code can be found at https://github.com/AI4Science-WestlakeU/diffphycon.

1 Introduction

Modeling the dynamics of complex physical systems is an important class of problems in science and engineering. Usually, we are not only interested in predicting a physical system's behavior but also injecting time-variant signals to steer its evolution and optimize specific objectives. This gives rise to the complex physical control problem, a fundamental task with wide applications, such as controlled nuclear fusion [10], fluid control [22], underwater devices [70] and aviation [47], among others.

Despite its importance, controlling complex physical systems efficiently presents significant challenges. It inherits the fundamental challenge of simulating complex physical systems, which are typically high-dimensional and highly nonlinear, as specified in Appendix A.1. Furthermore, observed control signals and corresponding system trajectories for optimizing a control model are typically far from the optimal solutions of the specific control objective. This fact poses a significant challenge of dilemma: *How to explore long-term control sequences beyond its training distribution*

[*]Equal contribution. [§]Work done as an intern at Westlake University. [†]Corresponding author.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

https://doi.org/10.52202/079017-0134

to seek near-optimal solutions while making the resulting system trajectory faithful to the dynamics of the physical system?

To tackle physical systems control problems, various techniques have been proposed, yet they fall short of addressing the above challenges. Regarding traditional control methods, the Proportional-Integral-Derivative (PID) control [33], is efficient but only suitable for a limited range of problems. Conversely, Model Predictive Control (MPC) [57], despite a wider range of applicability, suffers from high computational costs and challenges in global optimization. Recent advances in supervised learning (SL) [22, 24] and reinforcement learning (RL) [15, 46, 52], trained on system trajectories and control signals data, have demonstrated impressive performance in solving physical systems control problems. However, existing SL and model-based RL methods either fall into myopic failure modes [26] that fail to achieve long-term near-optimal solutions, or produce adversarial state trajectories [72] that violate the physical system's dynamics. The main reason may be that they treat the continuous evolution of dynamics from an iterative view, both lacking long-term vision and struggling in global optimization. See Appendix A.2 for more related work on physical system control.

In this work, we introduce Diffusion Physical systems Control (DiffPhyCon), a *new class* of method to address the physical systems control problem. We take an energy optimization perspective over system trajectory and control sequences across the whole horizon to implicitly capture the constraints inherent in system dynamics. We accomplish this through diffusion models, which are trained using system trajectory data and control sequences. In the inference stage, DiffPhyCon integrates simulation and control optimization into a unified energy optimization process. This prevents the generated system dynamics from falling out of distribution, and offers an enhanced perspective over long-term dynamics, facilitating the discovery of control sequences that optimize the objectives.

An essential aspect of physical systems control lies in its capacity to generate near-optimal controls, even when they may deviate significantly from the training distribution. We address this challenge with the key insight that the learned generative energy landscape can be *decomposed* into two components: a prior distribution representing the control sequence and a conditional distribution characterizing the system trajectories given the control sequence. Based on this insight, we develop a *prior reweighting* technique to subtract the effect of the prior distribution of control sequences, with adjustable strength, from the overall joint generative energy landscape during inference.

In summary, we contribute the following: **(1)** We develop DiffPhyCon, a novel generative method to control complex physical systems. By optimizing trajectory and control sequences jointly in the entire horizon by diffusion models, DiffPhyCon facilitates global optimization of long-term dynamics and helps to reduce myopic failure modes. **(2)** We introduce the prior reweighting technique to plan control sequences that are superior to those in the training set. **(3)** We demonstrate the effectiveness of our method on 1D Burgers' equation, 2D jellyfish movement control, and 2D high-dimensional smoke control tasks. On all three tasks, our method outperforms widely applied classical control methods and is also competitive with recent supervised learning and strong reinforcement learning baselines, particularly demonstrating advantages of DiffPhyCon in scenarios with partial observations and partial/indirect control. Notably, DiffPhyCon reveals the intriguing fast-close-slow-open pattern exhibited by the jellyfish, aligning with findings in the field of fluid dynamics [28]. **(4)** We generate a dataset of 2D jellyfish based on a CFD (computational fluid dynamic) software to mimic the movement of a jellyfish under control signals of its flapping behavior. To advance research in controlling complex physical systems, we release our dataset as a benchmark.

2 Background

2.1 Problem Setup

We consider the following complex physical system control problem:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{u}, \mathbf{w}) \quad \text{s.t.} \quad \mathcal{C}(\mathbf{u}, \mathbf{w}) = 0. \tag{1}$$

Here $\mathbf{u}(t, \mathbf{x}) : [0, \mathcal{T}] \times \Omega \mapsto \mathbb{R}^{d_\mathbf{u}}$ is the system trajectory $\{\mathbf{u}(t, \cdot), t \in [0, \mathcal{T}]\}$ defined on time range $[0, \mathcal{T}] \subset \mathbb{R}$ and spatial domain $\Omega \subset \mathbb{R}^D$, and $\mathbf{w}(t, \mathbf{x}) : [0, \mathcal{T}] \times \Omega \mapsto \mathbb{R}^{d_\mathbf{w}}$ is the external control signal for the physical system with dimension $d_\mathbf{w}$. $\mathcal{J}(\mathbf{u}, \mathbf{w})$ denotes the control objective. For example, $\mathcal{J}$ can be designed to measure the control performance towards a target state $\mathbf{u}^*$ with cost constraints: $\mathcal{J} \coloneqq \int \|\mathbf{u} - \mathbf{u}^*\|^2 \mathrm{d}\mathbf{x}\mathrm{d}t + \int \|\mathbf{w}\|^2 \mathrm{d}\mathbf{x}\mathrm{d}t$. $\mathcal{C}(\mathbf{u}, \mathbf{w}) = 0$ denotes the physical

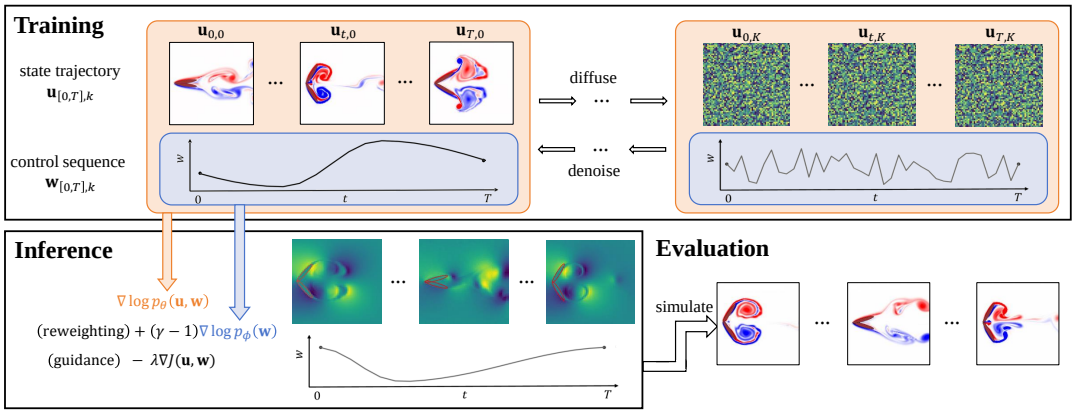


Figure 1: **Overview of DiffPhyCon**. The figure depicts the training (top), inference (bottom left), and evaluation (bottom right) of DiffPhyCon. Orange and blue colors respectively represent models learning the joint distribution $p_\theta(\mathbf{u}, \mathbf{w})$ and the prior distribution $p_\phi(\mathbf{w})$. Through prior reweighting and guidance, DiffPhyCon is capable of generating superior control sequences.

constraints. $\mathcal{C}(\mathbf{u}, \mathbf{w}) = 0$ can be specified either *explicitly* by a PDE dynamic which describes how the control signal $\mathbf{w}(t, \mathbf{x})$ drives the trajectory $\mathbf{u}(t, \mathbf{x})$ to evolve under boundary and initial conditions, or *implicitly* by control sequences and trajectory data collected from observation of the physical system. In the latter case, we may even have access to only partial trajectory data or engage in partial control. These situations collectively pose significant challenges to the physical system control task.

2.2 Preliminary: Diffusion Models

Diffusion models [19] are a class of generative models that learn data distribution from data. Diffusion models consist of two opposite processes: the forward process $q(\mathbf{x}_{k+1}|\mathbf{x}_k) = \mathcal{N}(\mathbf{x}_{k+1}; \sqrt{\alpha_k}\mathbf{x}_k, (1 - \alpha_k)\mathbf{I})$ to corrupt a clean data $\mathbf{x}_0$ to a Gaussian noise $\mathbf{x}_K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and the reverse parametrized process $p_\theta(\mathbf{x}_{k-1}|\mathbf{x}_k) = \mathcal{N}(\mathbf{x}_{k-1}; \mu_\theta(\mathbf{x}_k, k), \sigma_k\mathbf{I})$ to denoise from standard Gaussian $\mathbf{x}_K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\{\alpha_k\}_{k=1}^K$ is the variance schedule. To train diffusion models, [19] propose the DDPM method to minimize the following training loss for the denoising network $\boldsymbol{\epsilon}_\theta$, a simplification of the evidence lower bound (ELBO) for the log-likelihood of the data:

$$\mathcal{L} = \mathbb{E}_{k\sim U(1,K), \mathbf{x}_0 \sim p(x), \boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_k}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_k}\boldsymbol{\epsilon}, k)\|_2^2], \tag{2}$$

where $\bar{\alpha}_k := \prod_{i=1}^k \alpha_i$. $\boldsymbol{\epsilon}_\theta$ estimates the noise to be removed to recover data $\mathbf{x}_0$. During inference, iterative application of $\boldsymbol{\epsilon}_\theta$ from a Gaussian noise could generate a new sample $\mathbf{x}_0$ that approximately follows the data distribution $p(\mathbf{x})$. See Appendix A.3 for related work on diffusion models.

Notation. We use $\mathbf{v}_{[n,m]} = [\mathbf{v}_n, \cdots, \mathbf{v}_m]$ to denote a sequence of variables. We use $\mathbf{z}_{[0,T-1],k}$ to denote the hidden variable of $\mathbf{z}_{[0,T-1]}$ in a diffusion step k. For simplicity, we abbreviate $\mathbf{w}_{[0,T-1]}$, $\mathbf{u}_{[1,T]}$ as $\mathbf{w}$, $\mathbf{u}$. Concatenation of two variables is denoted via *e.g.*, $[\mathbf{u}, \mathbf{w}]$.

3 Method

In this section, we detail our method DiffPhyCon. In Section 3.1, we introduce our method including its training and inference. In Section 3.2, we further propose a prior reweighting technique to improve DiffPhyCon. The overview of DiffPhyCon is illustrated in Figure 1.

3.1 Generative Control by Diffusion Models

DiffPhyCon takes an energy optimization perspective to solve the problem Eq. (1), where PDE constraints can be modeled as a parameterized energy-based model (EBM) $E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c})$ which characterizes the distribution $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$ of $\mathbf{u}$ and $\mathbf{w}$ conditioned on conditions $\mathbf{c}$ by the correspondence $p(\mathbf{u}, \mathbf{w}|\mathbf{c}) \propto \exp(-E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c}))$. Lower $E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c})$, or equivalently higher $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$, means better satisfaction of the PDE constraints. Then the problem Eq. (1) can be converted to:

https://doi.org/10.52202/079017-0134

$$\mathbf{u}^*, \mathbf{w}^* = \underset{\mathbf{u},\mathbf{w}}{\operatorname{argmin}} \left[E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c}) + \lambda \cdot \mathcal{J}(\mathbf{u}, \mathbf{w}) \right], \tag{3}$$

where λ is a hyperparameter. This formulation optimizes $\mathbf{u}$ and $\mathbf{w}$ of all physical time steps simultaneously. The first term encourages the generated $\mathbf{w}$ and $\mathbf{u}$ to satisfy the PDE constraints $\mathcal{C}(\mathbf{u}, \mathbf{w}) = 0$. The second term guides optimization towards optimal objectives. The advantage of this optimization framework is that it tasks a global optimization on $\mathbf{u}$ and $\mathbf{w}$ of all times steps, which may obtain better solutions that are faithful to the dynamics of the physical system. Details about the effect of the hyperparameter λ are provided in Appendix L.

Training. To train E_θ, we exploit the diffusion model to estimate the gradient of E_θ. For convenience, we introduce a new variable $\mathbf{z}$ to represent the concatenation of $\mathbf{u}$ and $\mathbf{w}$ as $\mathbf{z} = [\mathbf{u}, \mathbf{w}]$. We use a denoising network $\boldsymbol{\epsilon}_\theta$ [19], which approximates $\nabla E_\theta(\mathbf{z}, \mathbf{c})$ [13], to learn the noise that should be denoised in each diffusion step $k = 1, \cdots, K$. The training loss of $\boldsymbol{\epsilon}_\theta$ is:

$$\mathcal{L} = \mathbb{E}_{k\sim U(1,K),(\mathbf{z},\mathbf{c})\sim p(\mathbf{z},\mathbf{c}),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_k}\mathbf{z} + \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}, \mathbf{c}, k)\|_2^2], \tag{4}$$

where $\boldsymbol{\epsilon}_\theta$ is conditioned on $\mathbf{c}$. Regarding training datasets, they could be generated by designing the conditions $\mathbf{c}$ and control sequences followed by a simulation when an explicit PDE form is available. Otherwise, they should be collected from observed pairs of control sequences and trajectories.

Control optimization. After the denoising network is trained, the Eq. (3) can be optimized by the Langevin sampling procedure as follows. We start from an initial sample $\mathbf{z}_K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and iteratively run the following process

$$\mathbf{z}_{k-1} = \mathbf{z}_k - \eta \left(\nabla_{\mathbf{z}} (E_\theta(\mathbf{z}_k, \mathbf{c}) + \lambda \mathcal{J}(\hat{\mathbf{z}}_k)) + \xi, \quad \xi \sim \mathcal{N}\big(\mathbf{0}, \sigma_k^2 \mathbf{I}\big), \right. \tag{5}$$

where σ_k^2 and η correspond to noise schedules and scaling factors used in the diffusion process, respectively. Here $\hat{\mathbf{z}}_k$ is the approximate noise-free $\mathbf{z}_0$ estimated from $\mathbf{z}_k$ by:

$$\hat{\mathbf{z}}_k = (\mathbf{z}_k - \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}_\theta(\mathbf{z}_k, \mathbf{c}, k))/\sqrt{\bar{\alpha}_k}. \tag{6}$$

We calculate $\mathcal{J}$ in (5) based on $\hat{\mathbf{z}}_k$ instead of directly using $\mathbf{z}_k$ because otherwise noise in $\mathbf{z}_k$ could bring errors to $\mathcal{J}$. Then ∇E_θ can be replaced by our trained denoising network $\boldsymbol{\epsilon}_\theta$ as follows:

$$\mathbf{z}_{k-1} = \mathbf{z}_k - \eta \left(\boldsymbol{\epsilon}_\theta(\mathbf{z}_k, \mathbf{c}, k) + \lambda \nabla_{\mathbf{z}} \mathcal{J}(\hat{\mathbf{z}}_k)) + \xi, \quad \xi \sim \mathcal{N}\big(\mathbf{0}, \sigma_k^2 \mathbf{I}\big) \right. \tag{7}$$

Iteration of this denoising process for $k = K, K-1, ..., 1$ yields a final solution $\mathbf{z}_0 = \{\mathbf{u}_{[1,T],0}, \mathbf{w}_{[0,T-1],0}\}$ for the optimization problem Eq. (3).

Guidance conditioning. In addition to the above introduced explicit guidance, conditioning is also widely used to guide sampling in diffusion models [20, 58]. When the control objective can be naturally expressed in a conditioning form, e.g., the generated trajectory $\mathbf{u}$ is required to coincide with a desired target $\mathbf{u}^*$, we can include $\mathbf{u} = \mathbf{u}^*$ as a condition in $\mathbf{c}$ such that the sampled trajectory $\mathbf{u}$ from diffusion models automatically satisfying $\mathbf{u} = \mathbf{u}^*$.

Overall, in our proposed DiffPhyCon framework, the control objective $\mathcal{J}$ can be optimized either using the explicit guidance $\nabla \mathcal{J}$ or guidance conditioning depending on the specific control objectives.

3.2 Prior Reweighting

Motivation. In physical systems control, a critical challenge lies in obtaining control sequences superior to those in training datasets, which often deviate significantly from achieving the optimal control objective. Although guidance of the control objective is incorporated in our diffusion model, generated control sequences are still highly influenced by the prior distribution of control sequences in training datasets. This inspires us to explore strategies to mitigate the effect of this prior, aiming to generate near-optimal control sequences.

We address this challenge with the key insight that the energy-based model $E(\mathbf{u}, \mathbf{w}, \mathbf{c})$ can be decomposed into two components: one is $E^{(p)}(\mathbf{w}, \mathbf{c})$ derived from the prior distribution $p(\mathbf{w}|\mathbf{c})$ of control sequences, and the other $E^{(c)}(\mathbf{u}, \mathbf{w}, \mathbf{c})$ representing the conditional probability distribution $p(\mathbf{u}|\mathbf{w}, \mathbf{c})$ of trajectories with respect to given control sequences. This decomposition has the following form

$$E(\mathbf{u}, \mathbf{w}, \mathbf{c}) = E^{(p)}(\mathbf{u}, \mathbf{c}) + E^{(c)}(\mathbf{u}, \mathbf{w}, \mathbf{c}), \tag{8}$$

by the corresponding decomposition of distribution $p(\mathbf{u}, \mathbf{w}|\mathbf{c}) = p(\mathbf{w}|\mathbf{c})p(\mathbf{u}|\mathbf{w}, \mathbf{c})$. We propose a *prior reweighting* technique, which introduces an adjustable hyperparameter $\gamma > 0$ as an exponential

of $p(\mathbf{w}|\mathbf{c})$, allowing for the tuning of the influence of this prior distribution. Then we have a reweighted version of $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$ as $p_\gamma(\mathbf{u}, \mathbf{w}|\mathbf{c}) = p(\mathbf{w}|\mathbf{c})^\gamma p(\mathbf{u}|\mathbf{w}, \mathbf{c})/Z$, which is also a probability distribution and can be further transformed to

$$p_\gamma(\mathbf{u}, \mathbf{w}|\mathbf{c}) = p(\mathbf{w}|\mathbf{c})^{\gamma-1} p(\mathbf{u}, \mathbf{w}|\mathbf{c})/Z, \tag{9}$$

where Z is a normalization constant. In particular, when $0 < \gamma < 1$, this approach is advantageous as it flattens the distribution $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$, thereby increasing the likelihood of sampling from low probability region of $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$, where the optimal solutions of the problem Eq. (3) probably lie.

Integrating Eq. (9) into Eq. (8), we have

$$E^{(\gamma)}(\mathbf{u}, \mathbf{w}, \mathbf{c}) = (\gamma - 1)E^{(p)}(\mathbf{w}, \mathbf{c}) + E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c}) - \log Z, \tag{10}$$

where $E^{(\gamma)}(\mathbf{u}, \mathbf{w}, \mathbf{c}) = -\log\left(p_\gamma(\mathbf{u}, \mathbf{w}|\mathbf{c})\right) + const$ is the reweighted energy-based model associated with $E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c})$ in Eq. (3), relying on the hyperparameter γ. Then the optimization problem Eq. (3) can be transformed to

$$\mathbf{u}^*, \mathbf{w}^* = \underset{\mathbf{u},\mathbf{w}}{\operatorname{argmin}} \left[E^{(\gamma)}(\mathbf{u}, \mathbf{w}, \mathbf{c}) + \lambda \cdot \mathcal{J}(\mathbf{u}, \mathbf{w}) \right]. \tag{11}$$

Optimization of this problem encourages sampling from the low likelihood region of $p(\mathbf{w}|\mathbf{c})$ while minimizing the control objective, which possesses the capability to generate control sequences that are more likely to be near-optimal than its degenerate version $\gamma = 1$ in the original optimization problem Eq. (3). The intuition of prior reweighting is illustrated in Figure 2.

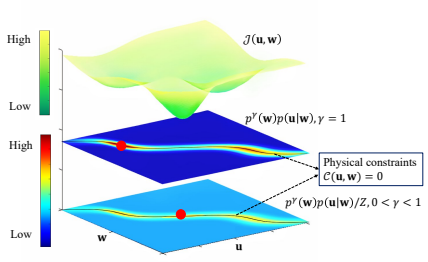


Figure 2: **Intuition of Prior Reweighting**. The top surface illustrates the landscape of $\mathcal{J}(\mathbf{u}, \mathbf{w})$, where the high-dimensional variables $\mathbf{u}$ and $\mathbf{w}$ are represented using one dimension. The middle and lower planes depict probability heatmaps for the reweighted distribution $p^\gamma(\mathbf{w})p(\mathbf{u}|\mathbf{w})/Z$. Adjusting γ from $\gamma = 1$ (middle plane) to $0 < \gamma < 1$ (lower plane), a better minimal of $\mathcal{J}$ (red dot in the lower plane) gains the chance to be sampled. This contrasts with the suboptimal red point in the middle plane highly influenced by the prior $p(\mathbf{w})$.

Training. To learn the reweighted energy $E^{(\gamma)}(\mathbf{u}, \mathbf{w}, \mathbf{c})$, we parameterize its gradient as a summation of two parts by taking the gradient of both sides of Eq. (10):

$$\begin{aligned}\nabla E^{(\gamma)}_{\phi,\theta}(\mathbf{u}, \mathbf{w}, \mathbf{c}) = (\gamma - 1)\nabla E^{(p)}_\phi(\mathbf{w}, \mathbf{c}) \\ + \nabla E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c}),\end{aligned}$$

where $E^{(p)}_\phi(\mathbf{w}, \mathbf{c})$ parameterizes the energy based model $E^{(p)}(\mathbf{w}, \mathbf{c})$ corresponding to $p(\mathbf{w}|\mathbf{c})$. Note that $\nabla \log Z$ vanishes here because it is a constant. Notice that $\nabla E_\theta(\mathbf{u}, \mathbf{w}, \mathbf{c})$ has already been trained by Eq. (4). $\nabla E^{(p)}_\phi(\mathbf{w}, \mathbf{c})$ can be trained similarly by the following loss function

$$\mathcal{L} = \mathbb{E}_{k\sim U(1,K),(\mathbf{w},\mathbf{c})\sim p(\mathbf{w},\mathbf{c}),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\phi(\sqrt{\bar{\alpha}_t}\mathbf{w} + \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}, \mathbf{c}, k)\|^2_2], \tag{12}$$

where $\boldsymbol{\epsilon}_\phi$ is the conditional denoising network that approximates $\nabla E^{(p)}_\phi(\mathbf{w}, \mathbf{c})$.

Control optimization. With both $\boldsymbol{\epsilon}_\theta$ and $\boldsymbol{\epsilon}_\phi$ trained, Eq. (11) can be optimized by running:

$$\mathbf{z}_{k-1} = \mathbf{z}_k - \eta(\boldsymbol{\epsilon}_\theta(\mathbf{z}_k, \mathbf{c}, k) + \lambda\nabla_{\mathbf{z}}\mathcal{J}(\hat{\mathbf{z}}_k)) + \xi_1, \quad \xi_1 \sim \mathcal{N}\big(0, \sigma_k^2\mathbf{I}\big) \tag{13}$$

$$\mathbf{w}_{k-1} = \mathbf{w}_{k-1} - \eta(\gamma - 1)\boldsymbol{\epsilon}_\phi(\mathbf{w}_k, \mathbf{c}, k) + \xi_2, \quad \xi_2 \sim \mathcal{N}\big(0, \sigma_k^2\mathbf{I}\big), \tag{14}$$

iteratively, where $\mathbf{z}_k = [\mathbf{u}_k, \mathbf{w}_k]$. The difference between this iteration scheme and Eq. (7) is that it uses an additional step to update $\mathbf{w}_k$ based on the predicted noise of $\boldsymbol{\epsilon}_\phi$. This guides $\mathbf{z}_k = [\mathbf{u}_k, \mathbf{w}_k]$ to move towards the reweighted distribution $p_\gamma(\mathbf{u}, \mathbf{w}|\mathbf{c})$ while aligning with the direction to decrease the objective by its guidance in the iteration of $\mathbf{z}_k$. The complete algorithm is present in Algorithm 1. Detailed discussion and results about how to set the hyperparameter γ are provided in Appendix L.

Theoretical Analysis. Consider a pair $[\mathbf{u}, \mathbf{w}]$ sampled using the prior reweighting technique with hyperparameter γ: $[\mathbf{u}, \mathbf{w}] \sim p_\gamma(\mathbf{u}, \mathbf{w}) = p(\mathbf{u}|\mathbf{w})p^\gamma(\mathbf{w})/C_\gamma$. Denote $\mathcal{J}^*$ as the global minimum

https://doi.org/10.52202/079017-0134

Algorithm 1 Inference for DiffPhyCon

Require Diffusion models $\boldsymbol{\epsilon}_\theta(\mathbf{z}_k, \mathbf{c}, k)$ and $\boldsymbol{\epsilon}_\phi(\mathbf{w}_k, \mathbf{c}, k)$, control objective $\mathcal{J}(\cdot)$, covariance matrix $\sigma_k^2 I$, control conditions $\mathbf{c}$, schedule $\bar{\alpha}_k$, hyperparameters λ, γ, K
Initialize optimization variables $\mathbf{z}_K = [\mathbf{u}_K, \mathbf{w}_K] \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $k = K, \ldots, 1$ **do**
  $\hat{\mathbf{z}}_k = (\mathbf{z}_k - \sqrt{1-\bar{\alpha}_k}\boldsymbol{\epsilon}_\theta(\mathbf{z}_k, \mathbf{c}, k))/\sqrt{\bar{\alpha}_k}$
  $\mathbf{z}_{k-1} = \mathbf{z}_k - \eta(\boldsymbol{\epsilon}_\theta(\mathbf{z}_k, \mathbf{c}, k) + \lambda \nabla_{\mathbf{z}} \mathcal{J}(\hat{\mathbf{z}}_k)) + \xi_1, \xi_1 \sim \mathcal{N}\big(0, \sigma_k^2 \mathbf{I}\big)$ // transition to next diffusion step
  $\mathbf{w}_{k-1} = \mathbf{w}_{k-1} - \eta(\gamma - 1)\boldsymbol{\epsilon}_\phi(\mathbf{w}_k, \mathbf{c}, k) + \xi_2, \xi_2 \sim \mathcal{N}\big(0, \sigma_k^2 \mathbf{I}\big)$ // prior reweighting
end for
return $\mathbf{u}^*, \mathbf{w}^* = \mathbf{z}_0$

of the control objective $\mathcal{J}$. Define $Q(\varepsilon)$ to be the "ϵ-optimal" solution set of $[\mathbf{u}, \mathbf{w}]$ such that $\mathcal{J}(\mathbf{u}, \mathbf{w}) - \mathcal{J}^* \leq (\epsilon)$, whose complement set is $Q(\varepsilon)^c$, and denote $\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w})$ as its indicator function, i.e. $\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) = 1$ if $[\mathbf{u}, \mathbf{w}] \in Q(\varepsilon)$; otherwise 0. Define Y to be the random variable of "whether use $\mathcal{J}$ as a guidance for sampling", namely,

$$p(Y|\mathbf{u}, \mathbf{w}) = \begin{cases} e^{-\mathcal{J}(\mathbf{u},\mathbf{w})}/Z, & Y = 1 \\ 1 - e^{-\mathcal{J}(\mathbf{u},\mathbf{w})}/Z, & Y = 0. \end{cases} \tag{15}$$

Consider $E(\gamma) = \mathbb{E}_{(\mathbf{u},\mathbf{w})\sim p_\gamma(\mathbf{u},\mathbf{w})}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w})|Y = 1]$, which indicates the expectation of getting an ϵ-optimal solution by using the prior reweighting technique with γ under the guidance of $\mathcal{J}$. Define

$$F(\gamma) = \frac{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w})p(Y = 1|\mathbf{u}, \mathbf{w}) \ln(p(\mathbf{w}))]}{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w})p(Y = 1|\mathbf{u}, \mathbf{w})]} - \frac{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u}, \mathbf{w})p(Y = 1|\mathbf{u}, \mathbf{w}) \ln(p(\mathbf{w}))]}{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u}, \mathbf{w})p(Y = 1|\mathbf{u}, \mathbf{w})]},$$

then we have the following theorem (please refer to Appendix B for its proof):

Theorem 3.1. *Assume $E(\gamma)$ is a smooth function, then the following hold:*

- *If $F(1) < 0$, there exists a $\gamma_- < 1$, s.t., $E(\gamma_-) > E(1)$;*
- *If $F(1) > 0$, there exists a $\gamma_+ > 1$, s.t., $E(\gamma_+) > E(1)$.*

Remark: Here $F(\gamma)$ can be interpreted as some kind of difference between "entropies" in $Q(\varepsilon)^c$ and $Q(\varepsilon)$. When $F(1) < 0$, it means that $Q(\varepsilon)^c$ has higher "entropies", implying that the training trajectories are far from optimal. As a result, we may need to flatten the distribution of training trajectories, which corresponds to using the prior reweighting technique with $\gamma < 1$. Since this is the most common case in real scenarios, we usually set $\gamma < 1$.

DiffPhyCon-lite. The introduction of the prior reweighting technique in DiffPhyCon involves training and evaluation of two models, thus bringing in additional computational cost. It is gratifying to note that we can balance the control performance and computational overhead of DiffPhyCon by adjusting the parameter γ. When $\gamma = 1$, the model $\boldsymbol{\epsilon}_\phi$ is not needed, and we denote this simplified version of DiffPhyCon as DiffPhyCon-lite.

4 Experiments

In this section, we aim to answer the following questions: (1) Can DiffPhyCon present superiority over traditional, supervised learning, and reinforcement learning methods for physical systems control? (2) Does the proposed prior reweighting technique help achieve better control objectives? (3) Could answers to (1) and (2) be generalized to more challenging partial observation or partial control scenarios? To answer these questions, we conduct experiments on three vital and challenging problems: 1D Burgers' equation, 2D jellyfish movement control, and 2D smoke control problems.

The following state-of-the-art control methods are selected as baselines. For the 1D Burgers' equation, we use (1) the classical and widely used control algorithm Proportional-Integral-Derivative (PID) [33] interacting with our trained surrogate model of the solver; (2) Supervised Learning method (SL) [24]; RL methods including (3) Soft Actor-Critic (SAC) [16] with offline and surrogate-solver versions; (4) Behaviour Cloning (BC) [49]; and (5) Behavior Proximal Policy Optimization (BPPO)

Table 1: **Best $\mathcal{J}_{\text{actual}}$ achieved in 1D Burgers's equation control.** Bold font denotes the best model, and underline denotes the second best model.

	PO-FC	FO-PC	PO-PC
PID (surrogate-solver)	-	0.09115	0.09631
SL	0.09752	<u>0.00078</u>	0.02328
SAC (surrogate-solver)	0.01577	0.03426	0.02149
SAC (offline)	0.03201	0.04333	0.03328
BC	0.02836	0.00856	0.00952
BPPO	0.02771	0.00852	<u>0.00891</u>
DiffPhyCon-lite (ours)	<u>0.01139</u>	**0.00037**	**0.00494**
DiffPhyCon (ours)	**0.01103**	**0.00037**	**0.00494**

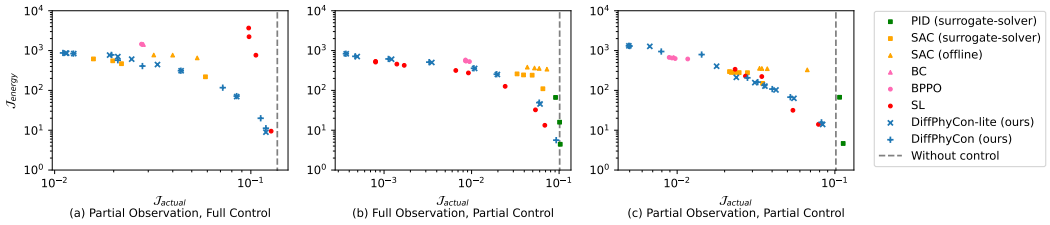


Figure 3: **Pareto frontier of $\mathcal{J}_{\text{energy}}$ vs. $\mathcal{J}_{\text{actual}}$ of different methods for 1D Burgers' equation.**

[73]. Specifically, the surrogate-solver version of SAC interacts with our trained surrogate model of the solver, while the offline version only uses given data. BC and BPPO are also in offline versions. For 2D jellyfish movement control, baselines include SL, SAC (offline), SAC (surrogate-solver), BC, BPPO, and an additional classical multi-input multi-output algorithm Model Predictive Control (MPC) [57]. PID is inapplicable to this data-driven task [2]. Detailed descriptions of baselines are provided in Appendix H and Appendix I.

4.1 1D Burgers' Equation Control

Experiment settings. The Burgers' equation is a governing law occurring in various physical systems. We consider the 1D Burgers' equation with the Dirichlet boundary condition and external force $w(t, x)$, which is also studied in [24, 42].

$$\begin{cases} \frac{\partial u}{\partial t} = -u \cdot \frac{\partial u}{\partial x} + \nu \frac{\partial^2 u}{\partial x^2} + w(t,x) & \text{in } [0,T] \times \Omega \\ u(t,x) = 0 & \text{on } [0,T] \times \partial\Omega \\ u(0,x) = u_0(x) & \text{in } \{t=0\} \times \Omega. \end{cases} \tag{16}$$

Here ν is the viscosity parameter, and $u_0(\mathbf{x})$ is the initial condition. Subject to Eq. (16), given a target state $u_d(x)$, the objective of control is to minimize the control error $\mathcal{J}_{\text{actual}}$ between u_T and u_d, while constraining the energy cost $\mathcal{J}_{\text{energy}}$ of the control sequence $w(t, x)$:

$$\mathcal{J}_{\text{actual}} \coloneqq \int_{\Omega} |u(T,x) - u_d(x)|^2 \mathrm{d}x, \quad \mathcal{J}_{\text{energy}} \coloneqq \int_{[0,T]\times\Omega} |w(t,x)|^2 \mathrm{d}t\mathrm{d}x. \tag{17}$$

To make the evaluation challenging, we select three different experiment settings that correspond to different real-life scenarios: partial observation, full control (PO-FC), full observation, partial control (FO-PC), and partial observation, partial control (PO-PC), which are elaborated on in Appendix D.2 and illustrated in Appendix C.1. These settings are challenging for classical control methods such as PID since they require capturing the long-range dependencies in the system dynamics. Note that the reported metrics in different settings are not directly comparable. In this experiment, DiffPhyCon uses the guidance conditioning (Section 3.1) to optimize $\mathcal{J}_{\text{actual}}$ and explicit guidance to optimize the energy cost, which is elaborated in Appendix D.3.

Results. In Table 1, we report results of the control error $\mathcal{J}_{\text{actual}}$ of different methods. It can be observed that DiffPhyCon delivers the best results compared to all baselines. Specifically, DiffPhyCon decreases $\mathcal{J}_{\text{actual}}$ of the best baseline by 30.1%, 52.6%, and 44.6% in the PO-FC, FO-PC, and PO-PC

https://doi.org/10.52202/079017-0134

Table 2: **Jellyfish movement dataset outline.**

training trajectories	test trajectories	resolution	#fluid features	trajectory length
30000	1000	128×128	3	40

settings respectively. From Table 1, DiffPhyCon and DiffPhyCon-lite show little performance gap. This is because, the prior distribution of w that our DiffPhyCon-lite learned is conditioned on both u_0 and u_T, which fully determines the optimal w. Therefore, $p(w|u_0, u_T)$ is intrinsically the optimal distribution and thus DiffPhyCon-lite can already deliver satisfactory performance.

To compare the ability of different methods to optimize $\mathcal{J}_{\text{actual}}$ with constrained energy cost $\mathcal{J}_{\text{energy}}$, we compare the Pareto frontiers of different methods in Figure 3. We vary the hyperparameter λ to control the tradeoff between $\mathcal{J}_{\text{actual}}$ and the energy cost since most baselines have this hyperparameter. As can be observed in Figure 3, the Pareto frontiers of DiffPhyCon are consistently among the best, achieving the *lowest* $\mathcal{J}_{\text{actual}}$ for most settings of the energy budget. Although SL performs well in full observation setting (b) where the system dynamics can be more easily predicted, it encounters difficulty in partial observation scenarios (a)(c). The results demonstrate DiffPhyCon's ability to generate near-optimal control sequences compared to baselines. More visualization results are provided in Appendix C.1. More results of evaluation are presented in Appendix K.1. For efficiency evaluation of training and test phases, please refer to Table 23 in Appendix K.4.

4.2 2D Jellyfish Movement Control

Experiment settings. This task is to control the movement of a flapping jellyfish with two wings in a 2D fluid field where fluid flows at a constant speed. The jellyfish is propelled by the fluid when its wings flap. Its moving speed and efficiency are determined by the mode of flapping. This task is an important source of inspiration for the design of underwater and aerial devices, and its challenges come from complex vortice behavior and fluid-solid coupling dynamics [54, 28]. For this task, fluid dynamics follows the 2D incompressible Navier-Stokes Equation:

$$\begin{cases} \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} - \nu \nabla^2 \mathbf{v} + \nabla p = 0 \\ \nabla \cdot \mathbf{v} = 0 \\ \mathbf{v}(0, \mathbf{x}) = \mathbf{v}_0(\mathbf{x}), \end{cases} \tag{18}$$

where $\mathbf{v}$ represents the 2D velocity of the fluid, and p represents the pressure, constituting the PDE state $\mathbf{u} = (\mathbf{v}, p)$. The initial velocity condition is $\mathbf{v}_0(\mathbf{x})$ and the kinematic viscosity is ν. We assume that each wing is rigid, so the jellyfish's boundary can be parameterized by the opening angle $\mathbf{w}_t$ of wings. Long-term movement of jellyfish usually presents a periodic flapping mode. Consequently, the control objective is to maximize its average moving speed $\bar{v}$ determined by the pressure of the fluid, under the energy cost constraint $R(\mathbf{w})$ and the periodic constraint $d(\mathbf{w}_T, \mathbf{w}_0)$ of the movement:

$$\mathcal{J} = -\bar{v} + \zeta \cdot R(\mathbf{w}) + d(\mathbf{w}_T, \mathbf{w}_0), \tag{19}$$

subject to Eq. (18) and the boundary condition that the velocity of fluid vanishes near the boundary. The hyperparameter ζ is set to be 1000. We evaluate in two settings: full observation, where the full state $\mathbf{u} = (\mathbf{v}, p)$ is observed; and partial observation, where only pressure is observed. This task is very challenging due to the complicated dynamics of fluid-solid interactions and vortices behaviour [54, 28], especially in the scenario of partial observation where the missing of $\mathbf{v}$ in Eq. (18) restricts the information available to generate well-informed control signals. Details of the experiment are provided in Appendix F.

Open-source Dataset Description. We use the Lily-Pad simulator [66] to generate a dataset describing the movement of jellyfish under the control of its flapping behavior. Statistics about the dataset are listed in Table 2. The feature of this dataset is that it contains a rich variation of vortices behavior and fluid-solid interaction dynamics, determined by violent changes in opening angles of wings and open-close phase ratio. It would serve as an important benchmark for studying complex physical system control problems. Details about the dataset are provided in Appendix E.

Results. Evaluation results are presented in Table 3. It can be seen that our method outperforms the baselines by a large margin in optimizing the control objective $\mathcal{J}$. At a cost of slightly increasing the control cost $R(\mathbf{w})$, control sequences generated by our method achieve a much faster average speed

Table 3: **2D jellyfish movement control results.** Bold font denotes the best model, and underline denotes the second best model.

	Full observation			Partial observation		
	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$
MPC	25.72	0.0112	109.17	-150.51	0.1791	329.59
SL	-76.94	0.1286	205.57	-102.98	0.1188	221.79
SAC (surrogate-solver)	-166.96	0.0069	18.14	-153.09	0.0057	158.82
SAC (offline)	-158.66	0.0069	165.58	-206.21	0.0058	211.96
BC	30.48	0.0629	32.44	20.08	0.0556	35.48
BPPO	<u>107.67</u>	0.0867	<u>-20.93</u>	<u>54.83</u>	0.0518	<u>-3.02</u>
DiffPhyCon-lite (ours)	95.04	0.0746	-20.47	2.92	0.0779	74.97
DiffPhyCon (ours)	**279.87**	0.2058	**-74.11**	**150.21**	0.1269	**-23.32**

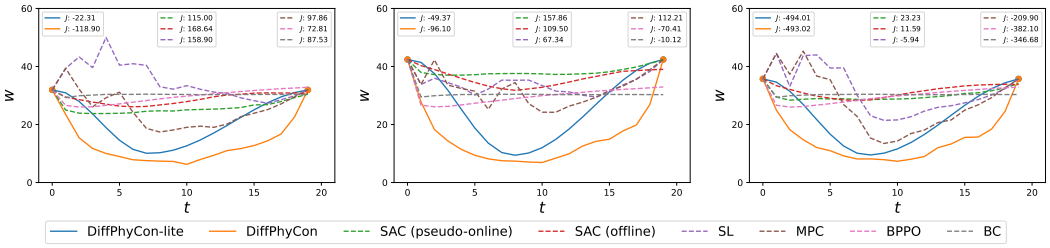


Figure 4: **Comparison of generated control curves of three test jellyfish.** The resulting control objective $\mathcal{J}$ for each curve is presented.

than baselines. In the full observation setting, the control objective achieved by DiffPhyCon is -53.18 lower than the best baseline BPPO, while the average speed exhibits an increment of 172.2 over it. This demonstrates that diffusion models are effective for this challenging control task by performing global optimization of trajectory and control sequences in a generative approach. Comparison between DiffPhyCon-lite and DiffPhyCon reveals that by flattening the prior distribution of control sequences, another significant improvement is further achieved. Even in the more challenging partial observation setting, DiffPhyCon still exhibits substantial advantages over existing methods. This reflects our method has a strong control capability under inadequate information. Configuration of the hyperparameter γ in DiffPhyCon and performance with respect to varying γ is presented in Figure 17 in Appendix L.1. Details about the hyperparameter λ can be found in Table 33 in Appendix L.2.

Figure 4 visualizes generated opening angle curves of different methods on three test jellyfish. Opening angle curves of DiffPhyCon-lite show an obvious fast-close-slow-open shape, which is proven to produce high speed in jellyfish movement [28]. The reason is that the fast closing of wings leads to a thrust from fluid in the early state, resulting in a long-term high speed, and followed by a slow opening to reduce resistance. While this mode of movement appears rarely in the training dataset, DiffPhyCon-lite could generate such control sequences for most test samples. This reflects that diffusion models under guidance are effective in optimizing the control objective. Furthermore, DiffPhyCon makes this mode of movement more aggressive, with sharper change of the opening angle in the beginning and end stage within a period. This provides strong evidence that reweighting the prior distribution of control sequences allows for flexible sampling over a flattened distribution, thus control sequences with low prior but good objectives are more likely to be sampled. The movement and the resulting fluid field of the jellyfish corresponding to the middle subfigure of Figure 4 controlled by DiffPhyCon is illustrated in Figure 5 and more examples are provided in Figure 10 in Appendix C.2. Conversely, opening angles obtained by baselines are inferior. The reason may be that they predict opening angles sequentially and hard to globally optimize the objective with three conflicting terms: average speed, $R(\mathbf{w})$, and $d(\mathbf{w}_T, \mathbf{w}_0)$. We further study such myopic failure mode of SAC in Appendix K.3. More comparisons about the variation of the weight ζ are listed in Table 20 and Table 21 in Appendix K.2. For efficiency evaluation of training and test phases, please refer to Table 24 in Appendix K.4. We also extend this experiment to a high-dimensional control signal setting, where the wings of the jellyfish are assumed to be soft. We find that our method is still competitive with baselines. Details about this evaluation are provided in Appendix M.

https://doi.org/10.52202/079017-0134

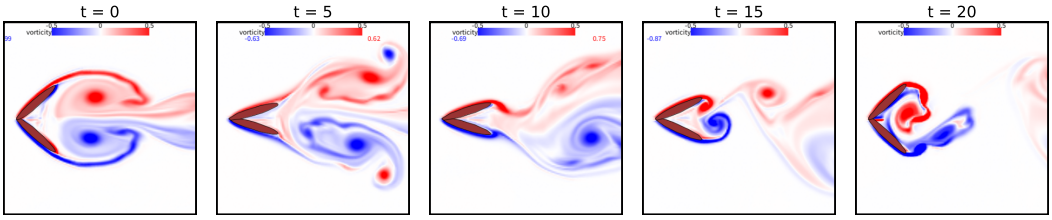


Figure 5: **Visualization of jellyfish movement and fluid field controlled by DiffPhyCon as in the middle subfigure of Figure 4.**

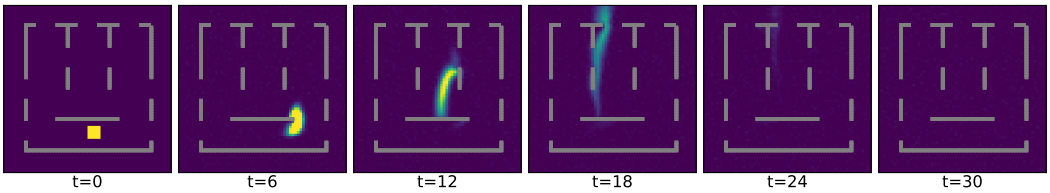


Figure 6: **Visualization of smoke density and fluid field dynamics controlled by DiffPhyCon.**

4.3 2D Smoke Indirect Control

Experiment settings. This task is to control smoke control in an incompressible fluid environment, following a similar (but more challenging) setup of [22]. Control forces were applied within a 64×64 grid flow field, excluding a semi-enclosed region, to minimize the smoke failing to pass through the top middle exit (seven exits in total). For illustration of our settings, please refer to Figure 14 in Appendix G. This high-dimensional indirect control problem involves managing 2D forces at approximately 1,700 grid points every time step, resulting in about 100,000 control variables across 32 time steps, making it highly challenging.

Results. Evaluation results are presented in Table 4. Our method still has significant advantages over baselines in minimizing the control objective. Furthermore, the prior reweighting technique achieves extra improvement over DiffPhyCon-lite. One test sample of smoke density and fluid field dynamics is illustrated in Figure 6. More visualization results of our method are presented in Figure 11 in Appendix G. These results demonstrate that DiffPhyCon is capable of controlling high dimensional physical systems even when control signals are indirectly applied to the system.

Table 4: **2D smoke movement control results.** Bold font denotes the best model, and underline denotes the second best model.

Method	$\mathcal{J} \downarrow$
BC	0.3085
BPPO	0.3066
SAC (surrogate-solver)	0.3212
SAC(offline)	0.6503
DiffPhyCon-lite (ours)	<u>0.2324</u>
DiffPhyCon (ours)	**0.2254**

5 Conclusion

In this work, we have introduced DiffPhyCon, a novel methodology for controlling complex physical systems. It generates control sequences and state trajectories by jointly optimizing the generative energy and control objective. We further introduced prior reweighting to enable the discovery of control sequences that diverge significantly from training. Through comprehensive experiments, we demonstrated our method's superior performance compared to classical, deep learning, and reinforcement learning baselines in challenging physical systems control tasks. We discuss limitation and future work in Appendix N and state social impact in O.

6 Acknowledgment

We thank Yuchen Yang for insightful discussions on theoretical analysis. We thank the anonymous reviewers for providing valuable feedback on our manuscript. We also gratefully acknowledge the support of Westlake University Research Center for Industries of the Future and Westlake University Center for High-performance Computing. The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding entities.

References

[1] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

[2] Karl Johan Åström and Tore Hägglund. The future of pid control. In *Digital Control – Past, present, and future of PID Control*, United States, 2000. Elsevier.

[3] Fan Bao, Min Zhao, Zhongkai Hao, Peiyao Li, Chongxuan Li, and Jun Zhu. Equivariant energy-guided SDE for inverse molecular design. In *The Eleventh International Conference on Learning Representations*, 2023.

[4] Gerben Beintema, Alessandro Corbetta, Luca Biferale, and Federico Toschi. Controlling rayleigh–bénard convection via reinforcement learning. *Journal of Turbulence*, 21(9-10):585–605, 2020.

[5] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for PDE modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

[6] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[7] Salva Rühling Cachay, Bo Zhao, Hailey James, and Rose Yu. Dyffusion: A dynamics-informed diffusion model for spatiotemporal forecasting. *arXiv preprint arXiv:2306.01984*, 2023.

[8] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *CoRR*, abs/2105.09506, 2021.

[9] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[10] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[11] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[12] Jie Ding, Min Wu, and Min Xiao. Nonlinear decoupling control with pi $^{\lambda}$ d $^{\mu}$ neural network for mimo systems. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–8, 2022.

[13] Yilun Du, Conor Durkan, Robin Strudel, Joshua B Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Sussman Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. In *International Conference on Machine Learning*, pages 8489–8510. PMLR, 2023.

[14] MA Elhawary. Deep reinforcement learning for active flow control around a circular cylinder using unsteady-mode plasma actuators. *arXiv preprint arXiv:2012.10165*, 2020.

[15] Amir-massoud Farahmand, Saleh Nabi, and Daniel N. Nikovski. Deep reinforcement learning for partial differential equation control. In *2017 American Control Conference (ACC)*, pages 3120–3127, 2017.

[16] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

 https://doi.org/10.52202/079017-0134

[17] Elie Hachem, Hassan Ghraieb, Jonathan Viquerat, Aurélien Larcher, and P Meliga. Deep reinforcement learning for the control of conjugate heat transfer. *Journal of Computational Physics*, 436:110317, 2021.

[18] Haoran He, Chenjia Bai, Kang Xu, Zhuoran Yang, Weinan Zhang, Dong Wang, Bin Zhao, and Xuelong Li. Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning. *Advances in neural information processing systems*, 36, 2024.

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[20] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[21] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv:2204.03458*, 2022.

[22] Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*, 2020.

[23] Benjamin Holzschuh, Simona Vegetti, and Nils Thuerey. Solving inverse physics problems with score matching. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[24] 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, pages 4504–4512, 2022.

[25] Allan Jabri, David Fleet, and Ting Chen. Scalable adaptive computation for iterative generation. *arXiv preprint arXiv:2212.11972*, 2022.

[26] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 9902–9915. PMLR, 17–23 Jul 2022.

[27] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *Proceedings of Machine Learning Research*, 162:9902–9915, 17–23 Jul 2022.

[28] Linlin Kang, An-Kang Gao, Fei Han, Weicheng Cui, and Xi-Yun Lu. Propulsive performance and vortex dynamics of jellyfish-like propulsion with burst-and-coast strategy. *Physics of Fluids*, 35(9), 2023.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023.

[31] Leon Lapidus and George F Pinder. *Numerical solution of partial differential equations in science and engineering*. John Wiley & Sons, 1999.

[32] A Larcher and E Hachem. A review on deep reinforcement learning for fluid mechanics: An update. *Physics of Fluids*, 34(11), 2022.

[33] Yun Li, Kiam Heong Ang, and G.C.Y. Chong. Pid control system analysis and design. *IEEE Control Systems Magazine*, 26(1):32–41, 2006.

[34] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[35] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[36] Jacques Louis Lions. *Optimal control of systems governed by partial differential equations*, volume 170. Springer, 1971.

[37] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[38] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. Compositional visual generation with composable diffusion models. In *European Conference on Computer Vision*, pages 423–439. Springer, 2022.

[39] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Transactions On Graphics (TOG)*, 23(3):449–456, 2004.

[40] Rajat Mittal and Gianluca Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.

[41] Keith W Morton and David Francis Mayers. *Numerical solution of partial differential equations: an introduction*. Cambridge university press, 2005.

[42] Saviz Mowlavi and Saleh Nabi. Optimal control of pdes using physics-informed neural networks. *Journal of Computational Physics*, 473:111731, 2023.

[43] Alex Nichol and Prafulla Dhariwal. Improved Denoising Diffusion Probabilistic Models. *ArXiv*, February 2021.

[44] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. *Proceedings of Machine Learning Research*, 162:16784–16804, 17–23 Jul 2022.

[45] Guido Novati, Siddhartha Verma, Dmitry Alexeev, Diego Rossinelli, Wim M Van Rees, and Petros Koumoutsakos. Synchronisation through learning for two self-propelled swimmers. *Bioinspiration & biomimetics*, 12(3):036001, 2017.

[46] Yangchen Pan, Amir-massoud Farahmand, Martha White, Saleh Nabi, Piyush Grover, and Daniel Nikovski. Reinforcement learning with function-valued action spaces for partial differential equation control. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3986–3995. PMLR, 10–15 Jul 2018.

[47] Aditya A Paranjape, Jinyu Guan, Soon-Jo Chung, and Miroslav Krstic. Pde boundary control for flexible articulated wings on a robotic aircraft. *IEEE Transactions on Robotics*, 29(3):625–640, 2013.

[48] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[49] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[50] Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Timo Ewalds, Andrew El-Kadi, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, Remi Lam, and Matthew Willson. Gencast: Diffusion-based ensemble forecasting for medium-range weather. *arXiv preprint arXiv:2312.15796*, 2023.

[51] Bartosz Protas. Adjoint-based optimization of pde systems with alternative gradients. *Journal of Computational Physics*, 227(13):6490–6510, 2008.

[52] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Réglade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of fluid mechanics*, 865:281–302, 2019.

 https://doi.org/10.52202/079017-0134

[53] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.

[54] Leif Ristroph and Stephen Childress. Stable hovering of a jellyfish-like flying machine. *Journal of The Royal Society Interface*, 11(92):20130992, 2014.

[55] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[56] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

[57] 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(5-6):1327–1349, 2021.

[58] Dule Shu, Zijie Li, and Amir Barati Farimani. A physics-informed diffusion model for high-fidelity flow field reconstruction. *Journal of Computational Physics*, 478:111972, 2023.

[59] Sabrine Slama, Ayachi Errachdi, and Mohamed Benrejeb. Neural adaptive pid and neural indirect adaptive control switch controller for nonlinear mimo systems. *Mathematical Problems in Engineering*, 2019, 2019.

[60] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[61] Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020.

[62] Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018.

[63] Pantelis R Vlachas, Georgios Arampatzis, Caroline Uhler, and Petros Koumoutsakos. Multiscale simulations of complex systems by learning their effective dynamics. *Nature Machine Intelligence*, 4(4):359–366, 2022.

[64] Marin Vlastelica, Tatiana Lopez-Guevara, Kelsey R Allen, Peter Battaglia, Arnaud Doucet, and Kim Stachenfeld. Diffusion generative inverse design. 2023.

[65] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

[66] Gabriel D Weymouth. Lily pad: Towards real-time interactive computational fluid dynamics. *arXiv preprint arXiv:1510.06886*, 2015.

[67] Tailin Wu, Takashi Maruyama, and Jure Leskovec. Learning to accelerate partial differential equations via latent global evolution. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, LA, USA, November 28 - December 9, 2022*, 2022.

[68] Tailin Wu, Takashi Maruyama, Long Wei, Tao Zhang, Yilun Du, Gianluca Iaccarino, and Jure Leskovec. Compositional generative inverse design. *arXiv preprint arXiv:2401.13171*, 2024.

[69] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy. *arXiv preprint arXiv:2403.03954*, 2024.

[70] Shuang Zhang, Xinyu Qian, Zhijie Liu, Qing Li, and Guang Li. Pde modeling and tracking control for the flexible tail of an autonomous robotic fish. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(12):7618–7627, 2022.

[71] Min Zhao, Fan Bao, Chongxuan Li, and Jun Zhu. Egsde: Unpaired image-to-image translation via energy-guided stochastic differential equations. *Advances in Neural Information Processing Systems*, 35:3609–3623, 2022.

[72] Qingqing Zhao, David B Lindell, and Gordon Wetzstein. Learning to solve PDE-constrained inverse problems with graph networks. *Proceedings of Machine Learning Research*, 162:26895–26910, 17–23 Jul 2022.

[73] Zifeng Zhuang, Kun Lei, Jinxin Liu, Donglin Wang, and Yilang Guo. Behavior proximal policy optimization. *arXiv preprint arXiv:2302.11312*, 2023.

 https://doi.org/10.52202/079017-0134

Appendix

Table of Contents

A Additional Related Work

A.1 Physical Systems Simulation

Complex physical systems simulation forms the foundation of systems control. While classical numerical techniques for simulating physical systems are renowned for their accuracy, they are often associated with significant computational expenses [41, 31]. Recently, neural network-based solvers show a significant advantage over classical solvers in accelerating simulations. They could be roughly divided into three primary classes: data-driven methods [34, 56, 48, 6, 67, 5, 30], Physics-Informed Neural Networks (PINNs) [53, 8, 65], and solver-in-the-loop methods [61, 63]. Most of them use an iterative horizontal prediction framework. Instead, we treat the system trajectory as a whole variable and use diffusion models to learn an explicit simulator conditioned on control sequences. A notable work is by [7], which introduces diffusion models for temporal forecasting. While both our work and [7]'s employ diffusion models, we tackle a different task of physical system control. Furthermore, we incorporate the control objective into the inference and introduce prior reweighting to tune the influence of the prior.

A.2 Physical Systems Control

For physical systems whose dynamics are described by PDEs, the adjoint methods [36, 39, 51] have been the most widely used approach for system control in the last decades. It is accurate but computationally expensive. Deep learning-based methods have emerged as a powerful tool for modeling physical systems' dynamics. Supervised learning (SL) [22, 24] trains parameterized models to directly optimize control using backpropagation through time over the entire trajectory. For example, [22] proposes a hierarchical predictor-corrector scheme to control complex nonlinear physical systems over long time frames. A more recent work proposed by [24] designs two stages which respectively learn the solution operator and search for optimal control. Different from these methods, we do not use the surrogate model, and learn both state trajectories and control sequences in an integrated way. Reinforcement learning (RL) [15, 46, 52] treats control signals as actions and learns policies to make sequential decisions. Particularly in the field of fluid dynamics [32], reinforcement learning has been applied to a multitude of specific problems including drag reduction [52, 14], conjugate heat transfer [4, 17] and swimming [45, 62]. But they implicitly consider physics information and sequentially make decisions. In contrast, we generalize the entire trajectories, which results in a global optimization with consideration of physical information learned by models.

 https://doi.org/10.52202/079017-0134

Recently, PINNs are also incorporated in PDE control [42], but they require an explicit form of PDE dynamics, while our method is data-driven and can deal with a broader range of complex physical system control problems without explicit PDE dynamics.

A.3 Diffusion Models

Diffusion models [19] have significantly advanced in applications such as image and text generation [11, 44], inverse design [68, 64], inverse problem [23], physical simulation [7, 50], and decision-making [27, 1, 18]. In particular, recent progress in robot control shows that diffusion models have significant advantages over existing reinforcement learning methods for action planning [9, 69]. Generating diverse yet consistent samples poses a challenge. For diversity, methods [38, 3, 71, 13] that integrate score estimates from various models have been effective. For consistency, guidance diffusion techniques [11, 20] have been utilized to generate condition-specific samples. Our approach differs by flattening the joint distribution to achieve better control by slightly expanding beyond the prior distribution range.

B Theoretical Analysis of Prior Reweighting

Proof of Theorem 3.1.

$$
\begin{aligned}
E(\gamma) =& \int \mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p_\gamma(u, w|Y=1) \mathrm{d}(\mathbf{u}, \mathbf{w}) \\
=& \int \mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) \frac{p(Y=1|\mathbf{u}, \mathbf{w}) p_\gamma(\mathbf{u}, \mathbf{w})}{p(Y=1)} \mathrm{d}(\mathbf{u}, \mathbf{w}) \\
=& \frac{\mathbb{E}_{(\mathbf{u},\mathbf{w})}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})]}{\mathbb{E}_{(\mathbf{u},\mathbf{w})}[p(Y=1|\mathbf{u}, \mathbf{w})]} \\
=& \frac{\mathbb{E}_{(\mathbf{u},\mathbf{w})\}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})]}}{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})] + \mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})]} \\
=& \frac{1}{1 + \frac{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})]}{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})]}}
\end{aligned}
$$

Define

$$
\begin{aligned}
G(\gamma) =& \frac{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})]}{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})]} \\
=& \frac{\mathbb{E}_w[\mathbb{E}_u[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})|\mathbf{w}]]}{\mathbb{E}_w[\mathbb{E}_u[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})|\mathbf{w}]]} \\
=& \frac{\int \mathbb{E}_u[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})|\mathbf{w}] p^\gamma(\mathbf{w}) \mathrm{d}\mathbf{w}}{\int \mathbb{E}_u[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u}, \mathbf{w}) p(Y=1|\mathbf{u}, \mathbf{w})|\mathbf{w}] p^\gamma(\mathbf{w}) \mathrm{d}\mathbf{w}}
\end{aligned}
$$

Then

$E(\gamma) = \frac{1}{1+\frac{1}{G(\gamma)}}$. Since $G(\gamma) > 0$, $E(\gamma)$ and $G(\gamma)$ have the same monotonicity.

$$G'(\gamma) = \frac{\int \mathbb{E}_u[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})|\mathbf{w}]p^\gamma(\mathbf{w})\ln(p(\mathbf{w}))\mathrm{d}\mathbf{w}\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})]}{(\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})])^2}$$

$$- \frac{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})]\int \mathbb{E}_u[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})|\mathbf{w}]p^\gamma(\mathbf{w})\ln(p(\mathbf{w}))\mathrm{d}\mathbf{w}}{(\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})])^2}$$

$$= \frac{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})\ln(p(\mathbf{w}))]\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})]}{(\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})])^2}$$

$$- \frac{\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})]\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})\ln(p(\mathbf{w}))]}{(\mathbb{E}_{\mathbf{u},\mathbf{w}}[\mathbb{I}_{Q(\varepsilon)^c}(\mathbf{u},\mathbf{w})p(Y=1|\mathbf{u},\mathbf{w})])^2}$$

By definition, $F(\gamma)$ is a positive multiple of $G'(\gamma)$, which implies our conclusion:

- If $F(1) < 0$, then $G'(1) < 0$ and thus $E(\gamma)$ decreases around 1. Hence, there exists $\gamma_- < 1$, s.t., $E(\gamma_-) > E(1)$, thus (i) holds;
- otherwise, for similar reason, (ii) holds.

□

Remark: Here $F(\gamma)$ can be interpreted as some kind of difference between "entropies" in $Q(\varepsilon)^c$ and $Q(\varepsilon)$. When $F(1) < 0$, it means that $Q(\varepsilon)^c$ has higher "entropies", implying that the training trajectories are far from optimal. As a result, we may need to flatten the distribution of training trajectories, which corresponds to using the prior reweighting technique with $\gamma < 1$. Since this is the most common case, we usually set $\gamma < 1$.

C Visualization Results

C.1 1D Burgers' Equation Visualization

We present more visualization results of our method and baselines under three settings: FOPC, POFC, and POPC in Figure 7, Figure 8 and Figure 9, respectively. Under each setting, we present the results of five randomly selected samples from the test dataset. The goal of control is to make the final state $\mathbf{u}_T$ ($T = 10$) close to the target state.

From these visualization results in Figure 7, Figure 8, and Figure 9, it can be observed that under the control of our DiffPhyCon, the system state could converge smoothly to the target state given different initial states, and the final state $\mathbf{u}_T$ always coincides with the target state. Furthermore, this observation is consistent under all three settings: FOPC, POFC, and POPC, implying that our DiffPhyCon is effective in addressing the partial observation and partial control challenges. In contrast, the baselines showed inferior results. Even the best baseline, BPPO, presents obvious mismatching with the target state on some samples.

https://doi.org/10.52202/079017-0134

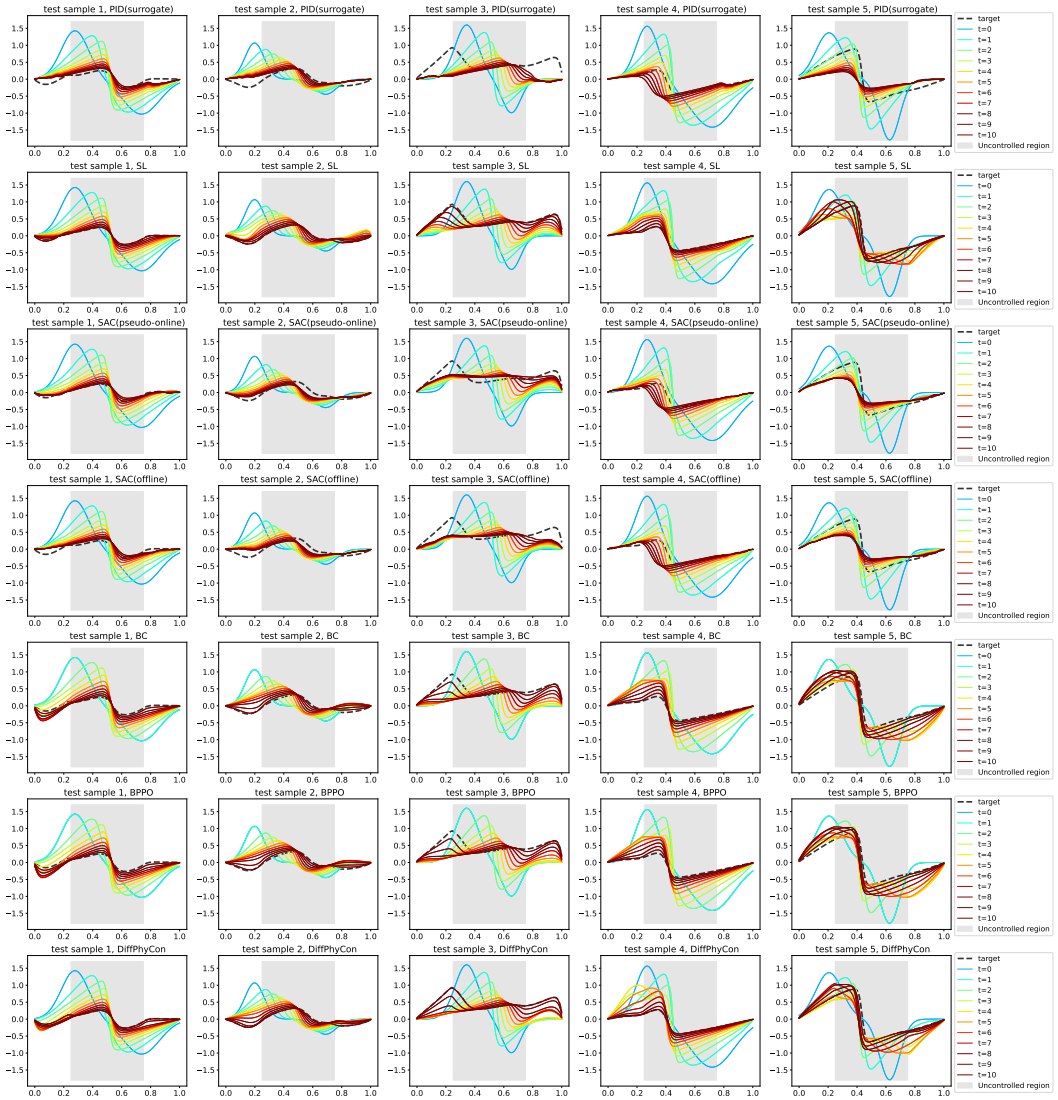


Figure 7: **Visualizations results of 1D Burgers' equation control under the FO-PC (full observation, partial control) setting**. The curve for the system state $\mathbf{u}_t$ of each time step $t = 0, \cdots, 10$ under control is plotted for our method (DiffPhyCon) and baselines. The x-axis is the spatial coordinate and the y-axis is the value of the system state.

https://doi.org/10.52202/079017-0134

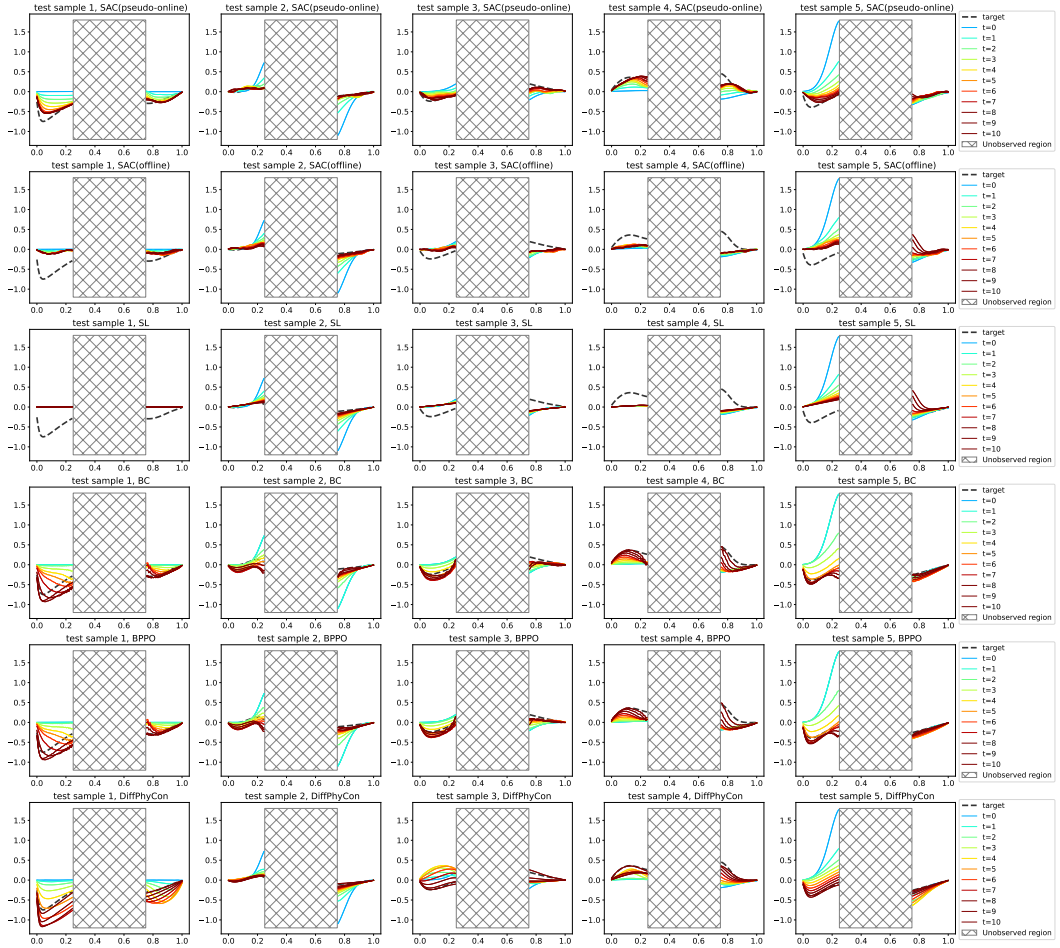


Figure 8: **Visualizations results of 1D Burgers' equation control under the PO-FC (partial observation, full control) setting**. The curve for the system state $\mathbf{u}_t$ of each time step $t = 0, \cdots, 10$ under control is plotted for our method (DiffPhyCon) and baselines. The x-axis is the spatial coordinate and the y-axis is the value of the system state.

https://doi.org/10.52202/079017-0134

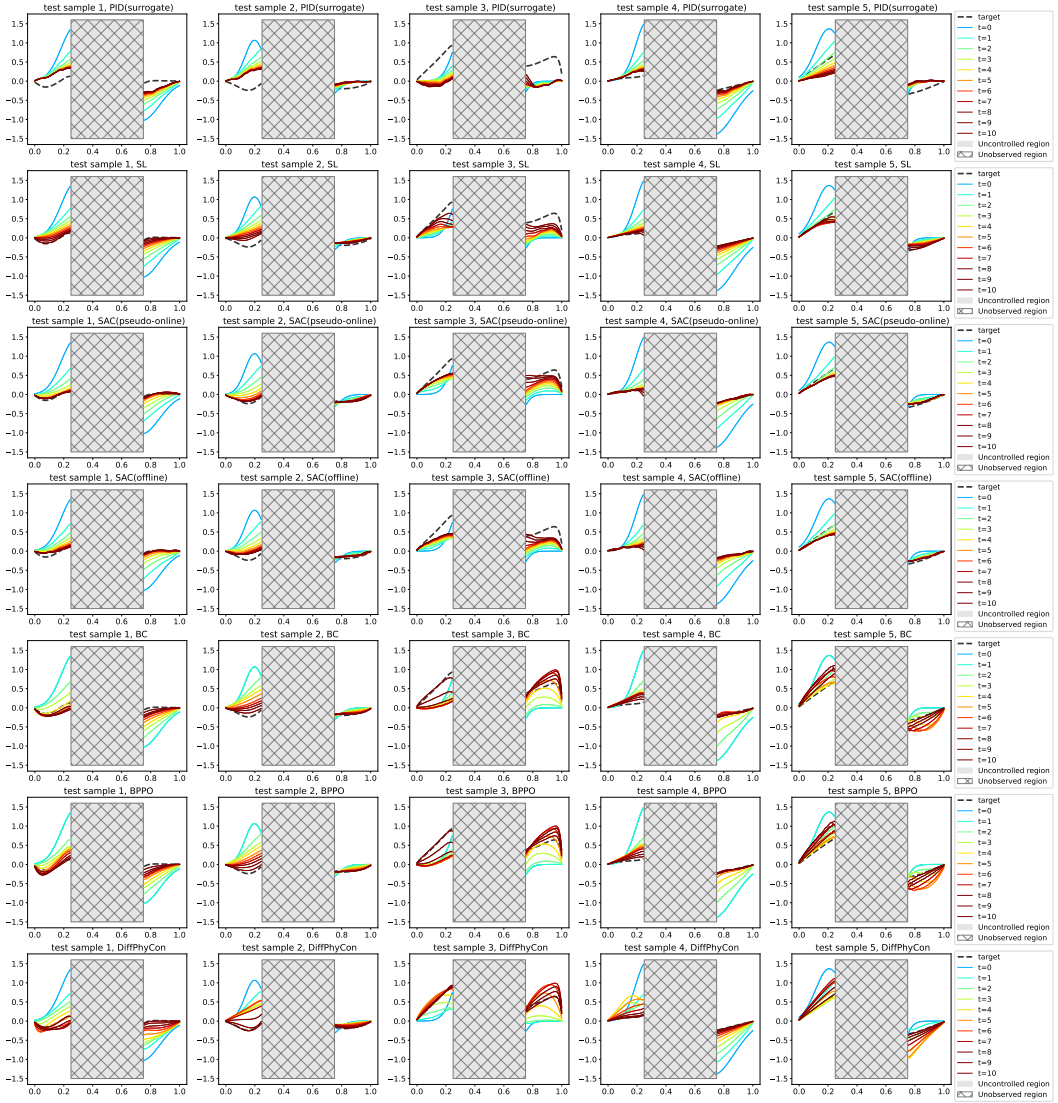


Figure 9: **Visualizations results of 1D Burgers' equation control under the PO-PC (partially observation, partially control) setting**. The curve for the system state $\mathbf{u}_t$ of each time step $t = 0, \cdots, 10$ under control is plotted for our method (DiffPhyCon) and baselines. The x-axis is the spatial coordinate and the y-axis is the value of the system state.

C.2 2D jellyfish control Visualization

We present more simulation results of our method in Figure 10. Each line represents an example from the test dataset. We plot five snapshots of boundary and fluid field for each example.

C.3 2D Smoke Control Visualization

We present fluid states and control signals generated by our method in Figure 11. Each line represents an example from the test dataset. We plot six snapshots for each example.

https://doi.org/10.52202/079017-0134

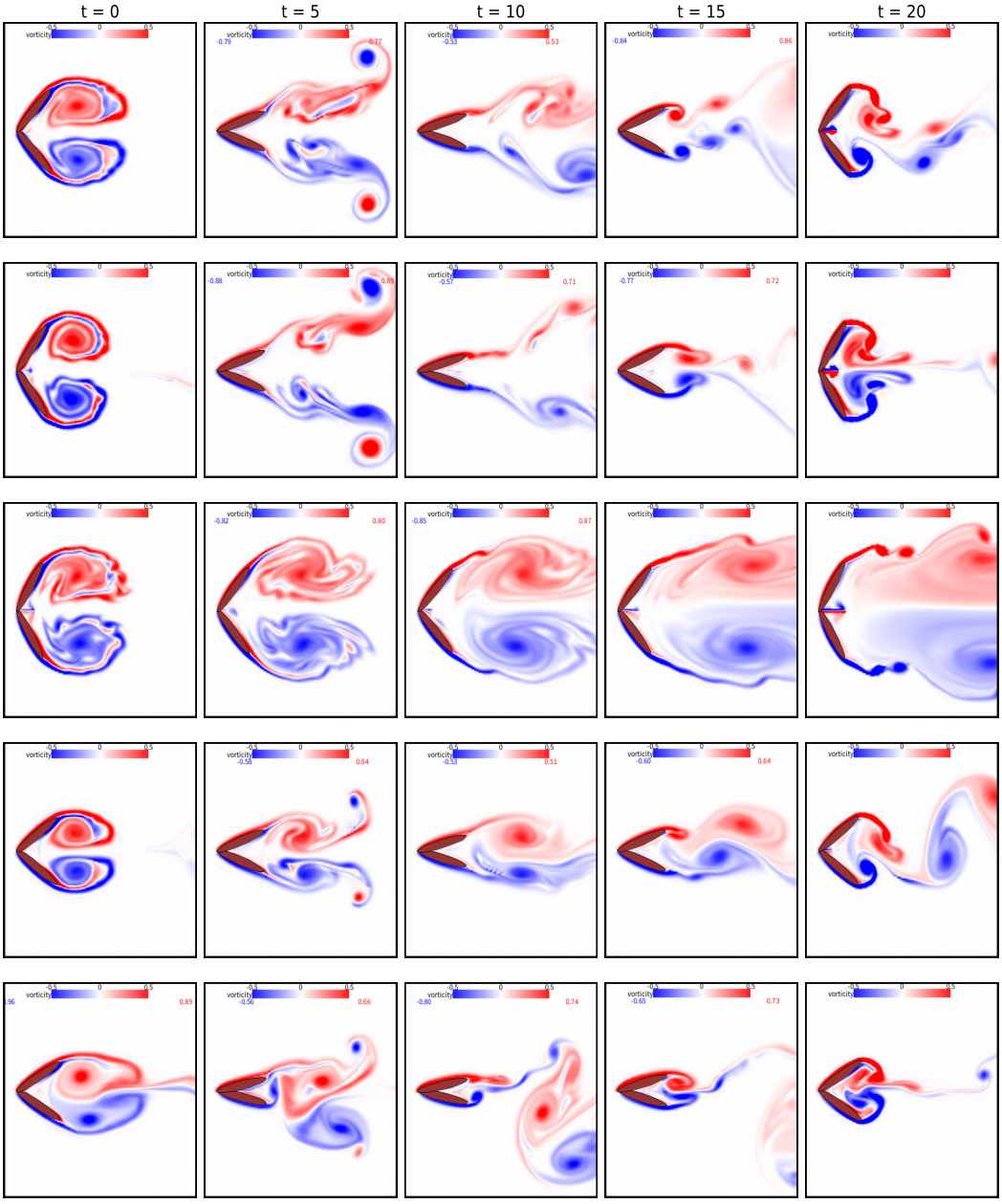


Figure 10: **More examples of 2D jellyfish simulation controlled by our method**.

https://doi.org/10.52202/079017-0134

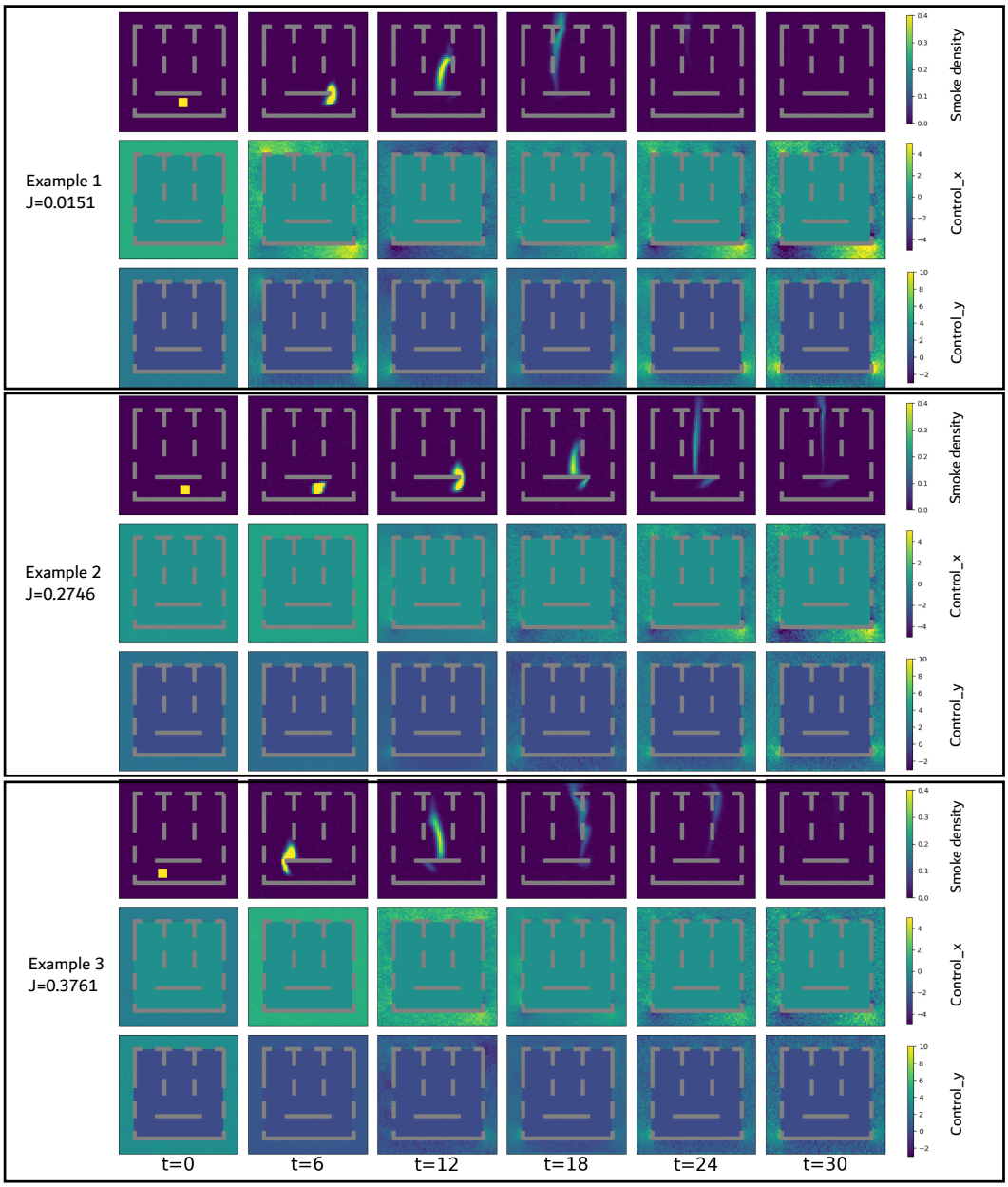


Figure 11: **Examples of 2D smoke control results by our method**. We present three randomly selected test examples. For each example, we show the generated smoke density map and control force fields in horizontal and vertical directions. Each row depicts six frames of movement. The smoke density in the first row corresponds to that in Figure 6.

https://doi.org/10.52202/079017-0134

D Additional Details for 1D Burgers' Equation Control

D.1 Data Generation

We use the finite difference method (called solver or ground-truth solver in the following) to generate the training data for the 1D Burgers' equation. Specifically, the initial value $\mathbf{u}_0(x)$ and the control sequence $\mathbf{w}(t,x)$ are both randomly generated, and then the states $\mathbf{u}(t,x)$ are numerically computed using the solver.

In the numerical simulation (using the ground-truth solver), a domain of $x \in [0,1],\ t \in [0,1]$ is simulated. The space is discretized into 128 grids and time into 10000 steps. However, in the dataset, only 10 time stamps are stored. For the control sequence $\mathbf{w}$, its refreshing rate is 0.1^{-1}, i.e., $\mathbf{w}(t,x),\ t \in [0.1k, 0.1(k+1)],\ k \in \{0,..,9\}$ does not change with t. Therefore, the data size of each trajectory is $[11, 128]$ for the state $\mathbf{u}$ and $[10, 128]$ for the control $\mathbf{w}$.

In all settings, the initial value $\mathbf{u}(0,x)$ is a superposition of two Gaussian functions $\mathbf{u}(0,x) = \sum_{i=1}^{2} a_i e^{-\frac{(x-b_i)^2}{2\sigma_i^2}}$, where a_i, b_i, σ_i are all randomly sampled from uniform distributions: $a_1 \sim U(0,2),\ a_2 \sim U(-2,0),\ b_1 \sim (0.2, 0.4),\ b_2 \sim (0.6, 0.8),\ \sigma_1 \sim U(0.05, 0.15),\ \sigma_2 \sim U(0.05, 0.15)$. Similarly, the control sequence $\mathbf{w}(x,t)$ is also a superposition of 8 Gaussian functions

$$\mathbf{w}(t,x) = \sum_{i=1}^{8} a_i e^{-\frac{(\mathbf{x}-b_{1,i})^2}{2\sigma_{1,i}^2}} e^{-\frac{(t-b_{2,i})^2}{2\sigma_{2,i}^2}}, \tag{20}$$

where each parameter is independently generated as follows: $b_{1,i} \sim U(0,1),\ b_{2,i} \sim U(0,1),\ \sigma_{1,i} \sim U(0.05, 0.2),\ \sigma_{2,i} \sim U(0.05, 0.2)$, while $a_1 \sim U(-1.5, 1.5)$ and for $i \geq 2$, $a_i \sim U(-1.5, 1.5)$ or 0 with equal probabilities. $\mathbf{u}(t,x),\ (t \neq 0)$ is then numerically simulated (using the ground-truth solver) given $\mathbf{u}(0,x)$ and $\mathbf{w}(t,x)$ based on Eq. (16). The setting of the dataset generation is based on a previous work [24]. We generated 90000 trajectories for the training set and 50 for the testing set. Each trajectory takes up 32KB space and the size of the dataset sums up to 2GB.

D.2 Experimental Setting

During inference, alongside the control sequence $\mathbf{w}(t,x)$, our diffusion model generates states $\mu(t,x)$, and some models produce surrogate states $\mu(t,x)$ when feeding the control $\mathbf{w}(t,x)$ into the corresponding surrogate model. However, our reported evaluation metric $\mathcal{J}_{\text{actual}}$ is always computed by feeding the control $\mathbf{w}(t,x)$ into the ground truth numerical solver to get $\mathbf{u}_{\text{g.t.}}(t,x)$ and computed following Eq. (17). Followings are three different settings of our experiments.

D.2.1 Partial Observation, Full Control

In realistic scenarios, the system is often unable to be observed completely. Generally speaking, it is impractical to place sensors *everywhere* in a system, so the ability of the model to learn from incomplete data is imperative. To evaluate this, we hide some parts of $\mathbf{u}$ in this setting and measure the $\mathcal{J}_{\text{actual}}$ of model control.

Specifically, $\mathbf{u}(t,x),\ x \in [\frac{1}{4}, \frac{3}{4}]$ is set to zero in the dataset during training and $\mathbf{u}_0(x),\ x \in [\frac{1}{4}, \frac{3}{4}]$ is also set to zero during testing. In this partial observation setting $\Omega = [1, \frac{1}{4}] \cup [\frac{3}{4}, 1]$. Since no information in the central $\frac{1}{2}$ space is ever known, the model does not know what will influence the control outcome of the unobserved states. Therefore, controlling the unobserved states is not a reasonable task and they are excluded from the evaluation metric.

This setting is particularly challenging not only because of the uncertainty introduced by the unobserved states but also the generation of the control in the central locations that implicitly affect the controlled $\mathbf{u}$ at $x \in \Omega$.

D.2.2 Full Observation, Partial Control

This is another setting of practical relevance, where only a fraction of the system can be controlled. The control sequence is enforced to be zero in the central locations of $x \in [\frac{1}{4}, \frac{3}{4}]$. Ω is still $[0,1]$, and $\mathcal{J}$ is evaluated on all of the observed states, though.

https://doi.org/10.52202/079017-0134

Some modifications to the dataset should be mentioned. The generation of the data involves first generating $\mathbf{w}$ as before, followed by setting the central $\frac{1}{2}$ of $\mathbf{w}$ to zero. To compensate for the decreased control intensity so that the magnitude of $\mathbf{u}$ can be roughly comparable to the full control setting, we double the magnitude of $\mathbf{w}$. During the evaluation, the output control sequence is also post-processed to be zero in $x \in [\frac{1}{4}, \frac{3}{4}]$.

It is worth noting that in this setting, even when the control energy is not limited at all, it is still challenging to find a perfect control since the model has to learn how to indirectly impose control on the central locations.

D.2.3 Partial Observation, Partial Control

The final setting is the combination of the previous two settings. Only $\Omega = [0, \frac{1}{4}] \cup [\frac{3}{4}, 1]$ is observed, controlled and evaluated.

It is worth noting that some models require accessing the current state to produce output. If the model interacts with the ground truth solver instead of a surrogate model, then the result would be unfairly good since the information of the unobserved states is leaked through the interaction.

D.3 Model

Since the training of models $\boldsymbol{\epsilon}_\phi \approx \nabla_{\mathbf{w}} \log p(\mathbf{w})$ and $\boldsymbol{\epsilon}_\theta \approx \nabla_{\mathbf{u},\mathbf{w}} \log p(\mathbf{u}, \mathbf{w})$ are essentially the same and the latter model is exactly DiffPhyCon-lite, we will introduce DiffPhyCon-lite first.

D.3.1 DiffPhyCon-lite

In general, DiffPhyCon-lite follows the formulation of [19] which is also described in the main text. The data of $\mathbf{u}$ and $\mathbf{w}$ is fed in as images of size (N_t, N_x) where N_t is the number of time steps (11 and 10 respectively) and N_x is the spatial grids (128). Since the two N_ts for $\mathbf{u}$ and $\mathbf{w}$ are inconsistent, we zero-pad them into the size of 16. Then, $\mathbf{u}$ and $\mathbf{w}$ are stacked as two channels and fed into the 2D DDPM model.

A 2D UNet $\boldsymbol{\epsilon}_\theta$ is used to learn to predict $\boldsymbol{\epsilon}$. It is structured into three main components: the downsampling encoder, the central module, and the upsampling decoder. The downsampling encoder is made up of four layers, each layer consisting of two ResNet blocks, one linear Attention block, and one downsampling convolution block. The central module also consists of two ResNet blocks and one linear Attention block. Each upsampling layer is the same as the downsampling layer except the downsampling block is replaced by the upsampling convolution block.

In our experiments, we found that the control result is best when learning the conditional probability distribution of $p(\mathbf{w}_{[0,T-1]}, \mathbf{u}_{[1,T-1]} \mid \mathbf{u}_0, \mathbf{u}_T)$. In summary, $\boldsymbol{\epsilon}_\theta$ takes in the current trajectory $\mathbf{u}$, control $\mathbf{w}$, step k, $\mathbf{u}_0$ and $\mathbf{u}_T$ as input, and predicts the noise of $\mathbf{u}$ and $\mathbf{w}$. Note that it is not trained to predict $\mathbf{u}_0$ and $\mathbf{u}_T$ which are used as a condition, but there are still model outputs at the corresponding locations for the data shape consistency across different design choices of DiffPhyCon-lite. The hyperparameters in different settings are listed in Table 5.

D.3.2 DiffPhyCon

In terms of implementation, DiffPhyCon is simply adding $\boldsymbol{\epsilon}_\phi(\mathbf{w})$ to $\boldsymbol{\epsilon}_\theta(\mathbf{u}, \mathbf{w})$ during inference as shown in Section 3.2, where $\boldsymbol{\epsilon}_\theta$ is the output of the denoising network in DiffPhyCon-lite while $\boldsymbol{\epsilon}_\phi$ is a new denoising network that is trained to generate $\mathbf{w}$ following the dataset distribution. Therefore, we only describe the model of $\boldsymbol{\epsilon}_\phi$ here.

$\boldsymbol{\epsilon}_\phi$ takes input of $\mathbf{w}, k$ as in the standard DDPM and $\mathbf{u}_0, \mathbf{u}_T$ as guidance conditioning. The output of $\boldsymbol{\epsilon}_\phi(\mathbf{w})$ is of the same shape as $\mathbf{w}$, so it can be treated as a network learning to sample from $p(\mathbf{w}) \coloneqq \int p(\mathbf{u}, \mathbf{w}) \mathrm{d}\mathbf{u}$ The output of $\boldsymbol{\epsilon}_\phi(\mathbf{w})$ at the locations of $\mathbf{u}$ is thus filled with zeros. The model hyperparameters are also listed in Table 5.

D.4 Training and Evaluation

Training During training, the $\mathbf{u}_0$ and $\mathbf{u}_d$ without noise are fed into the model and the model outputs at the corresponding locations are excluded from the loss. In the partial observation settings, the

Table 5: **Hyperparameters of the UNet architecture and training for the results of 1D Burgers' equation in Table 1**.

Hyperparameter name	Full observation Partial Control	Partial observation Full Control	Partial observation Partial Control
UNet $\boldsymbol{\epsilon}_\phi(\mathbf{w})$			
Initial dimension	32	32	32
Downsampling/Upsampling layers	4	4	4
Convolution kernel size	3	3	3
Dimension multiplier	$[1, 2, 4, 8]$	$[1, 2, 4, 8]$	$[1, 2, 4, 8]$
Resnet block groups	8	8	8
Attention hidden dimension	32	32	32
Attention heads	4	4	4
UNet $\boldsymbol{\epsilon}_\theta(\mathbf{u}, \mathbf{w})$			
Initial dimension	128	128	64
Downsampling/Upsampling layers	4	4	4
Convolution kernel size	3	3	3
Dimension multiplier	$[1, 2, 4]$	$[1, 2, 4, 8]$	$[1, 2, 4, 8]$
Resnet block groups	8	8	8
Attention hidden dimension	32	32	32
Attention heads	4	4	4
Training			
Training batch size	16	16	16
Optimizer	Adam	Adam	Adam
Learning rate	1e-4	1e-4	1e-4
Training steps	190000	170000	190000
Learning rate scheduler	cosine annealing	cosine annealing	cosine annealing
Inference			
Sampling iterations	1000	1000	1000
Intensity of energy guidance $\mathcal{J} = \int \|\mathbf{w}\|^2 \mathrm{d}x\mathrm{d}t$	0	0	0
Scheduler of energy guidance $\mathcal{J} = \int \|\mathbf{w}\|^2 \mathrm{d}x\mathrm{d}t$	cosine	cosine	cosine

unobserved data is invisible to the model during both training and testing as introduced in Appendix D.2. We simply pad zero in the corresponding locations of the model input and also exclude these locations in the training loss. Therefore, the model only learns the correlation between the observed states and control sequences. In the partial control setting, we train our DiffPhyCon on the dataset with control being zero in $x \in [\frac{1}{4}, \frac{3}{4}]$. In this way, the model naturally learns to output zero at the "non-controllable" locations.

We use the MSE loss to train the denoising UNets and other training hyperparameters are listed in Table 5.

Inference During inference, $\mathbf{u}_0$ and $\mathbf{u}_T$ are set to the target $\mathbf{u}_0$ and $\mathbf{u}_d$ so that the DDPM generates samples satisfying the physical constraint that is also conditioned on the target ($\mathbf{u}_d$) or the constraint ($\mathbf{u}_0$). In the partial observation setting, the $\mathbf{u}_0$ and $\mathbf{u}_T$ drawn from the testing set are all filled zero at the unobserved locations $x \in [\frac{1}{4}, \frac{3}{4}]$, which is the same as the data used to train the UNets. In the partial control scenarios,

During inference, we replace the denoising network's output $\boldsymbol{\epsilon}_\theta(\mathbf{u}, \mathbf{w})$ with $\boldsymbol{\epsilon}_\theta(\mathbf{u}, \mathbf{w}) + (\gamma - 1)\boldsymbol{\epsilon}_\phi(\mathbf{w})$. It is worth noting that $\boldsymbol{\epsilon}_\theta(\mathbf{u}, \mathbf{w})$ denoises u and $\mathbf{w}$ simultaneously while $\boldsymbol{\epsilon}_\phi(\mathbf{w})$ only denoises $\mathbf{w}$. In our experiments, we found that adding a schedule to the output of the $\mathbf{w}$ network is beneficial. The results in Table 1 are generated with a reverse Sigmoid schedule following

$$\boldsymbol{\epsilon}_k = \boldsymbol{\epsilon}_\theta(\mathbf{u}_k, \mathbf{w}_k, k, \mathbf{u}_0, \mathbf{u}_T) + \ (\gamma - 1)\beta_{K-k}\boldsymbol{\epsilon}_\phi(\mathbf{w}_k, k, \mathbf{u}_0, \mathbf{u}_T), \tag{21}$$

where β is defined as the noise schedule in [19]. The inference of DiffPhyCon-lite is simply setting $\gamma = 1$, which neglects the effect of the model $\boldsymbol{\epsilon}_\phi(\mathbf{w})$.

When trying to regulate the control energy, however, it is not as natural to learn a conditional model. Therefore, we use the external guidance $\mathcal{J} = \int \mathbf{w}(x, t)\mathrm{d}x\mathrm{d}t$ to produce cost-limited control sequences that are shown in Figure 3. The gradient of the external guidance is computed and added to $\boldsymbol{\epsilon}_k$ in Eq. (21). Note that we use the predicted clean sample $\hat{\mathbf{w}}_k$ and $\hat{\mathbf{u}}_k$ at the k-th step to compute $\nabla\mathcal{J}$ since they suffer less from being noisy and leading to deviated guidance. $\mathbf{u}_k$ and $\mathbf{w}_k$ are computed following $\mathbf{x}_0 \approx \hat{\mathbf{x}}_k = \frac{1}{\sqrt{\bar{\alpha}_k}}\mathbf{x}_k - \frac{\sqrt{1-\bar{\alpha}_k}}{\sqrt{\bar{\alpha}_k}}\boldsymbol{\epsilon}_k$ where $\mathbf{x}$ represents $\mathbf{u}$ or $\mathbf{w}$ as in [19]. In

https://doi.org/10.52202/079017-0134

our experiments, we use a cosine scheduling of the external guidance, and thus the final predicted noise would be $\boldsymbol{\epsilon}_k + \lambda \alpha_k \nabla \mathcal{J}$ where β_k is the noise schedule in [19] with the cosine schedule.

Evaluation After generating the trajectory $\mathbf{u}$ and control sequence $\mathbf{w}$, we feed the control sequence $\mathbf{w}$ into the ground-truth solver and simulate the final state $\mathbf{u}$ given the generated $\mathbf{w}$ and the initial condition $\mathbf{u}_0$ directly drawn from the testing dataset. The solver is the same as the one used in data generation in Appendix D.1. Finally, we compute $\mathcal{J}_{\text{actual}}$ following Eq. (17). In the partial observation setting, the MSE is computed only on the observed region, and in the control setting, the generated control will first be set to zero in the uncontrolled region before being fed into the solver.

E Jellyfish Movement Dataset

We use the Lily-Pad simulator [66] to generate the Jellyfish Movement dataset, which serves as a benchmark for physical system control research and also the dataset for our 2D evaluation task. Lily-Pad adopts the Immersed Boundary Method (IBM) [40] to simulate fluid-solid dynamics. The resolution of the 2D flow field is set to be 128×128. The flow field is assumed to be boundless in Lily-Pad. The head of the jellyfish is fixed at $(25.6, 64)$. Its two wings are represented by two identical ellipses, where the ratio between the shorter axis and the longer axis is 0.15. At each moment, the two wings are symmetric about the central horizontal line $y = 64$. For each wing, we sample $M = 20$ points along the wing to represent the boundary of the wing. The opening angle of the wings is defined as the angle between the longer axis of the upper wing and the horizontal line. It acts as the control sequence $\mathbf{w}$ in a 2D jellyfish control experiment.

Each trajectory starts from the largest opening angle and follows a cosine curve periodically with period $T^{'} = 200$. Trajectories differ in initial angle, angle amplitude, and phase ratio τ (the ratio between the closing duration and a whole pitching duration). For each trajectory, the initial angle $\mathbf{w}_0$ is generated as follows: first, sample a random angle, called mean angle $\mathbf{w}^{(m)} \in [20^\circ, 40^\circ]$, then sample a random angle amplitude $\mathbf{w}^{(a)} \in [10^\circ, \min(\mathbf{w}^{(m)}, 60^\circ - \mathbf{w}^{(m)})]$. The initial $\mathbf{w}_0$ is set as $\mathbf{w}_0 = \mathbf{w}^{(m)} + \mathbf{w}^{(a)}$. The phase ratio τ is randomly sampled from $[0.2, 0.8]$. The opening angle $\mathbf{w}_t$ of step t decreases from $\mathbf{w}^{(m)} + \mathbf{w}^{(a)}$ to $\mathbf{w}^{(m)} - \mathbf{w}^{(a)}$ as t grows from 0 to $\tau T^{'}$; then $\mathbf{w}_t$ increases from $\mathbf{w}^{(m)} - \mathbf{w}^{(a)}$ to $\mathbf{w}^{(m)} + \mathbf{w}^{(a)}$ as t grows from $\tau T^{'}$ to $T^{'}$. Afterwards, $\mathbf{w}_t$ varies periodically for $t > T^{'}$. The range of $\mathbf{w}_t$ is $[\mathbf{w}^{(m)} - \mathbf{w}^{(a)}, \mathbf{w}^{(m)} + \mathbf{w}^{(a)}] \subset [10^\circ, 60^\circ]$. For each trajectory, we simulate for 600 simulation steps, i.e., 3 periods. To save space, we only save the piece of trajectory from $T^{'} = 200$ to $3T^{'} = 600$ steps with step size 10 because the simulation from $t = 0$ to $T^{'} = 200$ is for initialization of the flow field. Then each trajectory is saved as a $\tilde{T} = (600 - 200)/10 = 40$ steps long sequence. An example of the simulated fluid field and the corresponding curve of opening angles are shown in Figure 12.

Besides the positions of the boundary points of wings and the opening angles $\mathbf{w}$, we also use another kind of image-like representation of the boundaries of wings as this representation contains spatial information that can be more effectively learned along with physical states (fluid field) by convolution neural networks. For each trajectory, this image-like boundary representation is compatible with physical states in shape. At each time step, boundaries of two wings are merged and then represented as a tensor of shape [3, 64, 64], where it has three features for each grid cell: a binary mask indicating whether the cell is inside a boundary (denoted by 1) or in the fluid (denoted by 0), and a relative position $(\Delta \mathbf{x}, \Delta y)$ between the cell center to the closest point on the boundary. For each trajectory, we save system states, opening angles, boundary points, boundary masks and offsets, and force data. They are specified as:

- system states $\mathbf{u}$: shape $[\tilde{T}, 3, 64, 64]$. For each step, we save the states of the fluid field consisting of velocity in $\mathbf{x}$ and y directions and pressure. To save space, we downsample the resolution from 128×128 to 64×64.
  - velocity: $[\tilde{T}, 2, 64, 64]$.
  - pressure: $[\tilde{T}, 1, 64, 64]$.
- opening angles $\mathbf{w}$: shape $[\tilde{T}]$. For each step, we save the opening angle in radians.
- boundary points: shape $[\tilde{T}, 2, M, 2]$. For each step, we save the boundary points on the upper and lower wings. Each wing consists of $M = 20$ points and each point consists

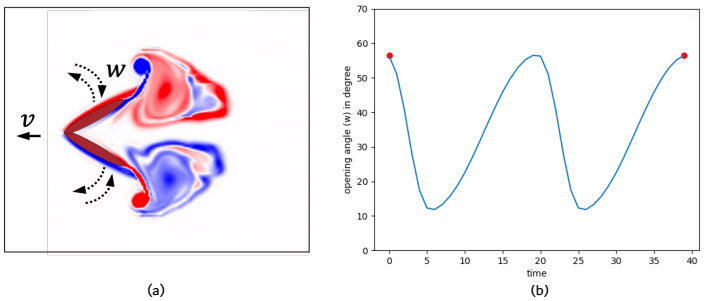


Figure 12: **Example of a flapping jellyfish in Lily-Pad simulator (a) and the corresponding curve of opening angle (b)**. The control signal is the opening angles $\mathbf{w}$ of the wings of a jellyfish.

of 2 coordinates. To make boundary points compatible with the downsampling of states, coordinates of $\mathbf{x}$ and y directions are shrunk to half $(64/128)$ of the original values.

- boundary mask and offsets b: $[\tilde{T}, 3, 64, 64]$. For each step, we save the mask of merged wings with half coordinates of boundary points and offsets in both $\mathbf{x}$ and y directions. The resolution is 64×64, compatible with that of the states.
  - mask: $[\tilde{T}, 1, 64, 64]$.
  - offsets: $[\tilde{T}, 2, 64, 64]$.
- force: shape $[\tilde{T}, 2]$. For each step, the simulator outputs the horizontal and vertical force from the fluid to the jellyfish. The horizontal force is regarded as a thrust to jellyfish if positive and a drag otherwise.

We generate $n = 30,000$ training trajectories and $n = 200$ testing trajectories. Trajectories differ in the above specified parameters $\mathbf{w}^{(a)}, \mathbf{w}^{(m)}$ and τ. Each trajectory occupies about 2MB of storage and the total dataset costs about 100GB.

F Additional Details for 2D Jellyfish Movement Control

F.1 Dataset Preparation

Based on our generated dataset in Appendix E, we prepare training samples for the 2D jellyfish movement control task as follows. We use sliding time windows that contain $T = 20$ successive time steps of states and boundaries as a sample, which corresponds to $T^{'} = 200$ original simulation steps and constitutes exactly a period of wing movement. In this way, each trajectory can produce 20 samples. Therefore, we get 6 million training samples in total. In each training sample, the initial and the final time steps share the same opening angle due to periodicity, which serves as the conditions for control. For each test trajectory, we select the opening angle of the jellyfish in the initial time and the initial states as the control condition for both the initial and final time and state initial condition.

F.2 Experimental Setting

F.2.1 Full Observation

In this setting, we assume all the states of the fluid field are observable. That is, both the velocity of x and y directions and pressure are available in all the time steps of the training dataset and the initial time of the testing dataset.

F.2.2 Partial Observation

In this setting, we assume only partial states are observed. A typical scenario in fluid simulation and control is that we can only observe pressure data while the velocity data is not easy to access. That is, only pressure is available in all the time steps of the training samples and the initial time of the testing samples, hence the state tensor is of shape $[\tilde{T}, 1, 64, 64]$. Notice that even if only pressure is available, we can still compute the force of fluid on the jellyfish and consequently the control objective because

https://doi.org/10.52202/079017-0134

force is fully determined by the shape of the jellyfish and pressure. The challenge of this partial observation setting is that the velocity variable v is missing in Eq. (18), which makes the traditional numerical solver no longer applicable to solve this physical system control problem. However, this challenge could be well addressed by our method since it could learn the relationship between control and pressure despite missing of the velocity data, and use the accessible control objective as guidance for flapping control.

F.3 Model

F.3.1 Architecture

We use a 3D U-Net as the backbone of our diffusion model, in both DiffPhyCon-lite and DiffPhyCon methods (detailed in the following subsection). In this paper, the architecture of the 3D U-Net we employed is inspired by [21]. To better capture temporal conditional dependencies, we modify the previous space-only 3D convolution into space-time 3D convolut ion. Notably, we did not perform any scaling on the temporal dimension during downsampling or upsampling. Specifically, our U-Net consists of three main modules: the downsampling encoder, the middle module, and the upsampling decoder. The downsampling encoder is composed of three layers, each incorporating two residual modules, one spatial attention module, one temporal attention module, and one downsampling module. The middle module consists of two residual modules, one spatial attention module, and one temporal attention module. Meanwhile, the upsampling decoder consists of four layers, each containing two residual modules, one spatial attention module, one temporal attention module, and one upsampling module. The input shape of our U-Net is [batch size, frames, channels, height, width]. During convolution, the operation is performed on the [frames, height, width] dimensions. The output shape follows the same structure. Further details are provided in Table 6.

Table 6: **Hyperparameters of 3D-Unet architecture**.

Hyperparameter name	Value
Kernel size of conv3d	(3, 3, 3)
Padding of conv3d	(1,1,1)
Stride of conv3d	(1,1,1)
Kernel size of downsampling	(1, 4, 4)
Padding of downsampling	(1, 2, 2)
Stride of downsampling	(0, 1, 1)
Kernel size of upsampling	(1, 4, 4)
Padding of upsampling	(1, 2, 2)
Stride of upsampling	(0, 1, 1)
attention heads	4

F.3.2 DiffPhyCon-lite

The DiffPhyCon-lite method learns the denoising network of the joint distribution $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$ where $\mathbf{u}$ is physical states, $\mathbf{w}$ is the opening angle, and the conditions $\mathbf{c}$ consist of the initial angle $\mathbf{w}_0$, the initial state $\mathbf{u}_0$ and the final angle $\mathbf{w}_T = \mathbf{w}_0$. We adopt the 3D U-Net as the backbone. To make the opening angle (of shape $[T]$), align with physical states (of shape $[T, 3, 64, 64]$ in full observation setting and $[T, 1, 64, 64]$ in partial observation setting) in shape, we expand the opening angle to shape $[T, 1, 64, 64]$ along spatial dimension by value copy. Besides, we also adopt the boundary mask and offsets representation, whose shape is $[T, 3, 64, 64]$, determined by the opening angles as an auxiliary model input because they contain explicit spatial features, which makes model learning more effective. Then states, boundary mask and offsets, and expanded opening angle are stacked along the channel dimension and we get a tensor of shape $[T, 7, 64, 64]$ in full observation setting or $[T, 5, 64, 64]$ in partial observation setting as the model input. The model output contains predicted noise of states and open angles. Thus its shape is $[T, 4, 64, 64]$ in the full observation setting or $[T, 2, 64, 64]$ in the partial observation setting, where the last channel corresponds to the predicted noise of opening angles and other channels correspond to predicted noise of states.

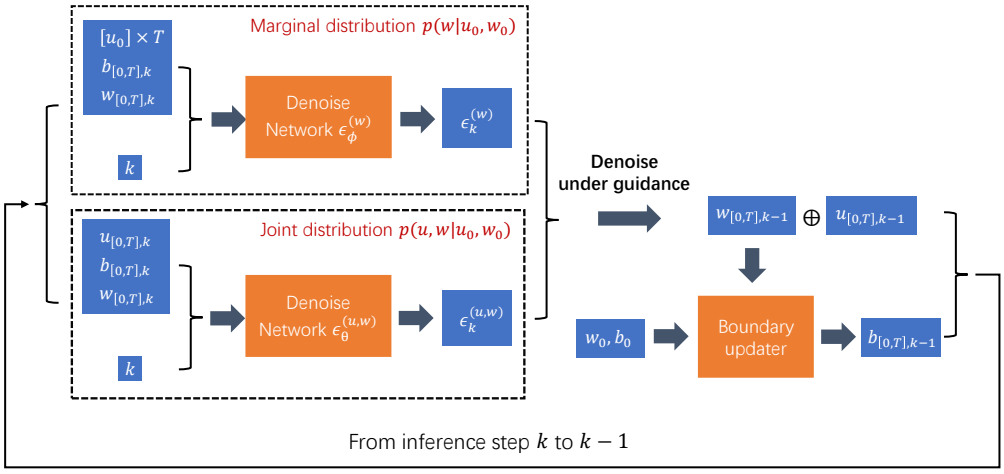


Figure 13: **Inference of our DiffPhyCon-lite and DiffPhyCon in the 2D experiment.**

Table 7: **Hyperparameters of network architecture and training for the 2D experiment**.

Hyperparameter name	full observation	partial observation
Batch size	16	16
Optimizer	Adam	Adam
Initial learning rate	0.001	0.001
Loss function	MSE	MSE

F.3.3 DiffPhyCon

DiffPhyCon learns the denoising network of the joint distribution $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$ and the marginal distribution $p(\mathbf{w}|\mathbf{c})$. The denoising network of $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$ is exactly the same as the one introduced in the DiffPhyCon-lite method. The denoising network of $p(\mathbf{w}|\mathbf{c})$ also adopts the 3D U-Net architecture. Its input size is the same as that of $p(\mathbf{u}, \mathbf{w}|\mathbf{c})$ in both full and partial observation settings. The difference is that the input states feature is replaced by the expansion of the initial state $\mathbf{u}_0$ along the time dimension by value copy. The output is the predicted noise of opening angles, whose shape is $[T, 1, 64, 64]$, no matter the full observation setting or partial observation setting.

F.4 Training, Inference, and Evaluation

Training. We use the MSE (mean squared error) between model prediction and the Gaussian noise as the loss function Eq. (4). The batch size is chosen as 16 and the training involves 200,000 iterations. The learning rate starts from 1×10^{-3} and multiplies a factor of 0.1 at the 50000th and 150000th iterations. Training details are provided in Table 7. The training is performed on two NVIDIA Tesla A100 GPUs with 80 GB memory for about 3 days.

Inference. The pipeline of inference is shown in Figure 13. Both diffused variables $\mathbf{u}_{[0,T]}$ and $\mathbf{w}_{[0,T]}$ are initialized from Gaussian prior and gradually denoised from denoising step $k = 1000$ to $k = 0$ based on denoising networks and guidance. Because we introduce the boundary mask and offsets as auxiliary inputs, the model input and output are not consistent in shape. Thus we introduce a surrogate model (shown as "Boundary updater" block in Figure 13) to update boundary mask and offsets $b_{[0,T],k}$ for each denoising k. Specifically, at each time step $t \in [0, T]$, $b_{t,k}$ is estimated by the initial boundary mask and offsets b_0, and the difference of opening angle $\mathbf{w}_{t,k} - \mathbf{w}_0$ from time step 0 to t, which is presented in the right part (after "Denoise under guidance") of Figure 13. Details about this surrogate model are presented in J.3. Notice that although this surrogate model is trained on noise-free data, we do not worry too much about its generalization to the noisy scalar $\mathbf{w}_{t,k}$ in

https://doi.org/10.52202/079017-0134

inference because the estimated $\mathbf{w}_{t,k}$ does not deviate from the normalized range of noisy free $\mathbf{w}_t$ too much.

Our method introduces two kinds of inference: DiffPhyCon-lite and DiffPhyCon. In DiffPhyCon-lite, we only use the denoising network of the joint distribution $p(\mathbf{u}, \mathbf{w}|\mathbf{u}_0, \mathbf{w}_0)$ for inference, while in DiffPhyCon, we use the additional denoising network of the marginal distribution $p(\mathbf{w}|\mathbf{u}_0, \mathbf{w}_0)$ together with that of the joint distribution for inference. These two branches are plotted in the left part of Figure 13, where the notation $[\mathbf{u}_0] \times T$ means expand initial state $\mathbf{u}_0$ (of shape $[3, 64, 64]$) along time dimension by value copy to form a tensor of shape $[T, 3, 64, 64]$.

As for guidance, we use a surrogate force model to approximate the force of fluid on jellyfish. This model is detailed in Subsection J.2. In denoising step k, its input consists of two parts: the first one is the noise-free state $\hat{\mathbf{u}}_{[0,T]}$ estimated from $\mathbf{u}_{[0,T],k}$ by Eq. (6); the second one is the noise-free boundary mask and offsets $\hat{b}_{[0,T],k}$ estimated from noise-free $\hat{\mathbf{w}}_{[0,T]}$ by the surrogate model to update boundaries, where $\hat{\mathbf{w}}_{[0,T]}$ is also estimated from $\mathbf{w}_{[0,T],k}$ as in Eq. (6). The model output is force. Here we only use the horizontal force. Notice that the force could be computed via the surrogate force model no matter whether states are fully or partial observation in that force is irrelevant to the velocity of the fluid. The control objective $\mathcal{J}$ in Eq. (19) is computed as a summation of force and $R(\hat{\mathbf{w}}_{[0,T]})$. We fix $\zeta = 1000$ as a default setting in Eq. (19) because this value can achieve a balance between scales of the average speed and the regularizer $R(\mathbf{w})$. We also study the Pareto performance of varying ζ in Table 3. Then the gradients of the objective $\mathcal{J}$ in terms of $\hat{\mathbf{u}}_{[0,T]}$ and $\hat{\mathbf{w}}_{[0,T]}$ are computed and used in guidance. For DiffPhyCon-lite, these gradients are substracted from $[\mathbf{u}_{[0,T],k}, \mathbf{w}_{[0,T],k}]$ to generate $[\mathbf{u}_{[0,T],k-1}, \mathbf{w}_{[0,T],k-1}]$. For DiffPhyCon, an additional term of noise $(\gamma - 1)\boldsymbol{\epsilon}_\phi$ predicted from the denoising network of the marginal distribution $p(\mathbf{w}|\mathbf{u}_0, \mathbf{w}_0)$ should also be subtracted, as shown in the upper left part of Figure 13. The effect of this term is controlled by the scale of the hyperparameter γ, as studied in Appendix L. The inference is performed on one NVIDIA Tesla-V100 GPU with 32 GB memory for about 3 hours for 50 testing samples.

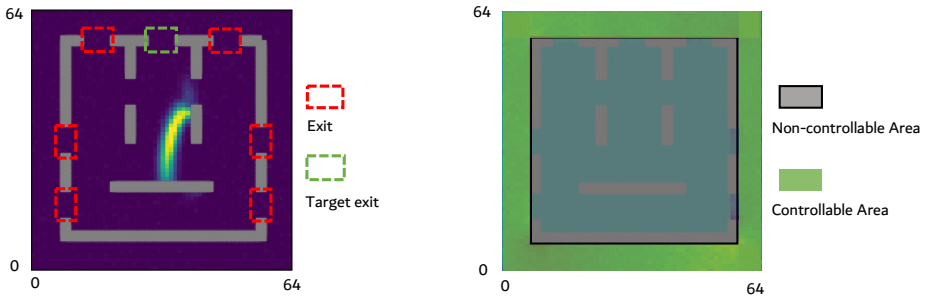
Evaluation. The inference outputs opening angles $\mathbf{w}_{[0,T]}$ of $T = 20$ steps for 50 testing samples. In simulation, for each testing sample, the ground-truth first $T = 20$ steps of the opening angles (which corresponds to 200 simulation steps) are directly input to the Lily-Pad simulator for the reason of generating initial states u_0 of fluid, which is followed by the predicted control sequences of opening angles (interpolated to 200 steps of opening angles). The simulator outputs the horizontal force of fluid on the jellyfish for each simulation step. Finally, average speed $\bar{v}$, energy cost $R(\mathbf{w})$, and objective $\mathcal{J}$ are computed as metrics. The average speed $\bar{v} = \frac{1}{T}\int_0^T v_t \mathrm{d}t \approx v_0 + \frac{1}{T}\sum_{t=1}^{T-1}(T-t)F_t$ $\bar{v}$, where v_0 is the initial speed and F_t is the horizontal thrust from the fluid. The mass of the jellyfish is assumed to be 1. The energy cost term $R(\mathbf{w}) = \sum_{t=1}^{T-1}(\mathbf{w}_{t+1} - \mathbf{w}_t)^2$, where the control sequence $\mathbf{w} = (\mathbf{w}_1, \cdots, \mathbf{w}_T)$ represents the predicted opening angles. The periodic term $d(\mathbf{w}_0, \mathbf{w}_T) = \max(|\mathbf{w}_T - \mathbf{w}_0| - \boldsymbol{\epsilon}, 0)$ is the constraint of periodic opening angles with a small threshold $\epsilon = 0.01$.

G Additional Details for 2D Smoke Control

G.1 Dataset Preparation

We use the Phiflow solver [22] to generate the incompressible fluid with an indirect control dataset. The resolution of the 2D flow field is set to be 64×64. The flow field is set to be boundless in Phiflow. We placed obstacles and absorbing areas in the middle of the fluid domain. In Figure 14, we illustrate the locations of the exits, obstacles, and controllable areas. The specific positions of the obstacles and absorbing areas are detailed in Table 8 & Table 9.

At the beginning of each trajectory, we set the horizontal velocity component v_x of the entire flow field to zero and the vertical component v_y to an upward velocity of 1.0. Additionally, we randomly initialize a square smoke patch with dimensions of 4×4 in the area below the horizontal obstacles, specifically within the coordinates (11:50, 11:14). In the trajectories we designed, the smoke is required to make four turns. We randomly determine the positions where the smoke will turn, and calculate the smoke's horizontal (v_x) and vertical (v_y) velocity components, assuming a constant magnitude of velocity. By adjusting the parameters, we observed that when the v_x added at the turning points is sampled from a Gaussian distribution $\mathcal{N}(mv_x, m^2v_x^2/16)$ and v_y is sampled from



(a) Locations of exits and obstacles (b) Locations of controllable area

Figure 14: **2D smoke indirect control task**. There are seven exits in total and the top middle one is the target exit (a). The control signals are only allowed to apply to peripheral regions (b). The control objective is to minimize the proportion of smoke failing to pass through the target exit.

Table 8: **Obstacle Positions**

Category	Position
Bottom	(8:56,8:9)
Left	(8:9,8:12) (8:9,20:28) (8:9,36:56)
Right	(56:57,8:12) (56:57,20:28) (56:57,36:56)
Up	(8:12,56:57) (20:28,56:57) (36:44,56:57) (52:56,56:57)
Inside Obstacles	(24:25,32:40) (24:25,48:56) (40:41,32:40) (40:41,48:56) (20:44, 20:21)

Table 9: **Absorb Area**

Category	Absorb Area
Left	(0:8, 11:21) (0:8, 27:37)
Right	(56:64, 11:21) (56:64, 27:37)
Up	(11:21, 56:64) (27:37, 56:64) (43:53, 56:64)

a Gaussian distribution $\mathcal{N}(6v_y, 9v_y^2/4)$, where is m sampled from a uniform distribution $U(2, 7)$, the success rate of the smoke navigating through is approximately 53.87% in the training set and 52.86% in the testing set, which meets our requirements. During the motion of the smoke, at turning points, the velocity in the central region is set to the velocity from the previous moment, while the velocity in the surrounding regions is assigned by randomly sampling from the distributions $\mathcal{N}(v_{turn}, {v_{turn}}^2/100)$. At non-turning points, the velocity in the central region remains the velocity from the previous moment, but the velocity in the surrounding areas is adjusted to the previous velocity plus noise sampled from $\mathcal{N}(0, 0.01)$. This procedure ensures that each control parameter varies at every moment and remains distinct from others at any given time. Such variability is beneficial for the subsequent training and generation of control parameters.

We duplicated the density field into two versions: the original density field and the set-zero density field. The original density field is used for model training and remains unaffected by the absorbing areas. In contrast, the set-zero density field is employed to calculate the amount of smoke passing through each bucket. When set-zero density is present in the absorb areas, we sum up and record this density. Once the recording is complete, we reset the set-zero density in the absorb areas to zero to prevent the double-counting of emitted smoke. Ultimately, we document the quantity of smoke emitted from each bucket at every moment.

https://doi.org/10.52202/079017-0134

Our experimental data records the velocity and density of the entire flow field at each moment, as well as the velocity of the peripheral flow field (control field) after noise addition. We generated 40,000 training trajectories and 500 test trajectories, with each trajectory approximately 10 MB in size. Given a total of 40,500 trajectories, the entire dataset is estimated to be around 400 GB in size.

G.2 Experimental Setting

In smoke control, we only consider the full observation and full control setting. Namely, all fluid features, including smoke density, horizontal and vertical velocities are observable; and all green areas shown in Figure 14 are controllable.

G.3 Model

G.3.1 Architecture

Similar to 2D jellyfish control, we use a 3D U-Net as the backbone of our diffusion model, in both DiffPhyCon-lite and DiffPhyCon methods (detailed in the following subsections). Details of the architecture are same to Table 6.

G.3.2 DiffPhyCon-lite

Except for three fluid features, we use the accumulated proportion of smoke passing through the target exit as another feature. To make this feature (of shape $[T]$), align with fluid features (of shape $[T, 3, 64, 64]$) in shape, we expand it to shape $[T, 1, 64, 64]$ along spatial dimension by value copy. The horizontal and vertical control signals are represented by a tensor of shape $[T, 2, 64, 64]$, where uncontrollable positions are filled with zeros. Then these four state features and control signals are stacked along the channel dimension and we get a tensor of shape $[T, 6, 64, 64]$ as the model input. The model output contains predicted noise of the state features and control signals. Thus its shape is also $[T, 6, 64, 64]$.

G.3.3 DiffPhyCon

The denoising network of $p(\mathbf{w}|\mathbf{c})$ also adopts the 3D U-Net architecture. The input and output size is $[T, 2, 64, 64]$.

G.4 Training, Inference, and Evaluation

Training. The batch size is chosen as 6. The number of iterations and learning rate schedule are same to 2D jellyfish control. The training is performed on two NVIDIA Tesla A6000 GPUs with 48 GB memory for about 2 days.

Inference. The pipeline of inference is similar to 2D jellyfish control as shown in Figure 13. The difference is that we do not involve any surrogate model in this task. In each sampling step, we use the estimated proportion of smoke failing to exit through the target exit as guidance. The inference is performed on one NVIDIA Tesla A6000 GPU with 48 GB memory for about 3 hours for 50 testing samples.

Evaluation. The inference outputs the control force $\mathbf{w}_{[0,T]}$ of $T = 64$ steps for 50 testing samples. In simulation, for each testing sample, the initial state and the generated control signals are input to the simulator. The simulator outputs the proportion of smoke failing to exit through the target exit for the final time step. Then the proportions are averaged over 50 test samples.

H 1D Burgers' Equation Control Baselines

H.1 PID

Proportional Integral Derivative (PID) [33] control is a versatile and effective control method widely used in various real-world control scenarios. It operates by utilizing the difference (error) between the desired target and the current state of a system. PID control is often considered the go-to option for many control problems due to its simplicity and usefulness. However, despite its popularity, PID

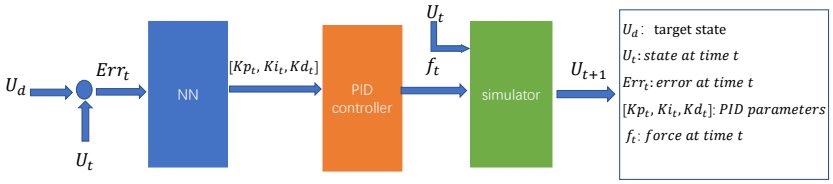


Figure 15: **The architecture of ANN-PID.** To use MIMO PID controller to control U_t to U_d, we train a neural-network-based PID parameter planner to output MIMO PID parameters based on Err_t, then use the PID controller to output the control sequence f_t.

control does encounter certain challenges, such as parameter adaptation and limitations when applied to Single Input Single Output (SISO) systems. In our specific context, the 1D Burgers' Equation Control problem presents a Multiple Input Multiple Output (MIMO) control scenario, which makes it infeasible to directly employ PID control to regulate the Burgers' equation. Inspired by the early works [59, 12] using a neural network as a PID parameter adapter, we have integrated deep learning with PID control to tackle the MIMO control problem. As shown in Figure 15, ANN(artificial neural network) PID uses a neural network as a PID parameter adapter to output multiple sets of PID parameters and do multiple sets of SISO PID control.

The neural network to output PID parameters comprises two 1D convolutional layers, 2 fully connected layers, and 4 corresponding activation layers. We use the $L1$ loss of the current state and target state as training loss and the Adam optimizer [29] to train the model. Detailed information can be found in Table 10.

Table 10: **Hyperparameters of network architecture and training for ANN-PID**.

Hyperparameter name	Full observation	Partial observation
Kernel size of conv1d	3	3
Padding of conv1d	1	1
Stride of conv1d	1	1
Activation function	Softsign	Softsign
Batch size	16	16
Optimizer	Adam	Adam
Learning rate	0.0001	0.0001
Loss function	MAE	MAE

As PID itself is a SISO control method, ANN-PID uses a neural network to get multiple sets of PID parameters to do multiple SISO PID controls for MIMO control in the Burgers' equation. But here, ANN-PID requires the dimensions of inputs and outputs to be the same, so it can only cope with full observation, full control control problems, and partial observation, partial control control problems. Besides, the ANN-PID controller has 2 training setups, including directly interacting with the solver and interacting with the 1D surrogate model in Appendix J.

H.2 SAC

The Soft Actor-Critic (SAC) algorithm [16] is a cutting-edge reinforcement learning method. Conceptualized as an improvement over traditional Actor-Critic methods, SAC distinguishes itself by introducing an entropy regularization term into the loss function, which encourages the policy to explore more efficiently by maximizing both the expected cumulative reward and the entropy of the policy itself.

Compared with Deep Deterministic Policy Gradient (DDPG) algorithm [35, 46], SAC's entropy regularization encourages more effective exploration and prevents early convergence to suboptimal policies, a limitation often seen with DDPG's deterministic approach. Additionally, SAC's twin Q-networks mitigate the overestimation bias that can affect DDPG's value updates, leading to more stable learning. The automatic tuning of the temperature parameter in SAC further simplifies the delicate balance between exploration and exploitation, reducing the need for meticulous hyperparameter

https://doi.org/10.52202/079017-0134

Table 11: **Hyperparameters of 1D SAC.** The full observation partial control, partial observation full control, and partial observation partial control settings share the same hyperparameters.

Hyperparameter name	Value
Hyperparameters for 1D Burgers' equation control:	
Discount factor for reward	0.5
Target smoothing coefficient	0.05
Learning rate of critic loss	0.0003
Learning rate of entropy loss	0.003
Learning rate of policy loss	0.003
Training batch size	8192
Number of episodes	1500
Number of model updates per simulator step	50
Value target updates per step	15
Size of replay buffer	1000000
Number of trajectories interacted with the environment per step	1
Number of layers of critic networks	3
Number of hidden dimensions of critic networks	4096
Number of layers of the policy network	5
Number of hidden dimensions of the policy network	4096
Activation function	ReLU
Clipping's range of policy network's standard deviation output	$\left[e^{-20}, e^{2}\right]$

adjustments. Consequently, these features render SAC generally more sample-efficient and robust, particularly in complex and continuous action spaces.

During training, experience for training is stored in a replay buffer and sampled randomly to update the networks. All data in the training set are in the replay buffer at the beginning. For offline SAC, the replay buffer is unchanged, while online SAC alternates between collecting experience by interacting with the environment and updating the networks with the replay buffer. And offline SAC only uses the surrogate model trained with the training set instead of the real environment to collect experience. The policy network is updated to maximize the expected return, considering both the Q-value and the entropy term. The critic networks are updated to minimize the distance between their Q-value predictions and the target Q-values. SAC also employs a target critic network for the critic networks, which are slowly updated with the weights of the main critic network to stabilize training. For the inference, SAC uses the policy network to determine the action by selecting the action with the highest probability.

In practice, to help the system approximate the target state accurately and quickly, we need to include the distance between the states of every time step and the target state in the reward. So the reward function of time step t, state u_t, target state u_T and action $\mathbf{w}_t$ here is defined as

$$r(t, \mathbf{u}_t, y_T, \mathbf{w}_t) = -\int_{\Omega} |\mathbf{u}_t - \mathbf{u}_d|^2 \mathrm{d}\mathbf{x} - \alpha \int_{\Omega} |\mathbf{w}_t|^2 \mathrm{d}\mathbf{x},$$

where Ω is the space domain and α is the weight of energy. We take the Adam optimizer [29] to train the networks and update the temperature parameter. The detailed values of hyperparameters are provided in Table 11.

H.3 Supervised Learning

The paper [24] proposes a supervised-learning-based control algorithm that takes a neural operator as a surrogate model to solve control problems. It contains two stages. In the first stage, we take a neural operator to learn the physical constraint as Appendix J. The three CNNs respectively reconstruct u, reconstruct $\mathbf{w}$ and learn the transition from u_t to u_{t+1}. More details are in Appendix J. In the second stage, these three neural networks are used as surrogate models to calculate the gradient of the objective function with respect to the control input. We consider the control $\mathbf{w}$ as a learnable parameter and update it with the gradient.

To enhance the accuracy, we adopt the LBFGS optimizer [37], which is more accurate while slower than the Adam optimizer. We record the hyperparameters of the second stage in Table 12.

Table 12: **Hyperparameters of the second stage of the 1D supervised learning method**.

Hyperparameter name	Value
Hyperparameters for 1D Burgers' equation control: (full observation partial control)	
Learning rate of $\mathbf{w}$ updating	0.1
Number of epochs	300
Weight of objective function loss	500
Weight of reconstruction loss	0.03
Termination tolerance on first order optimality of LBFGS optimizer	4×10^{-7}
Termination tolerance on parameter changes LBFGS optimizer	4×10^{-7}
Hyperparameter name	Value
Hyperparameters for 1D Burgers' equation control: (partial observation partial/full control)	
Learning rate of $\mathbf{w}$ updating	0.1
Number of epochs	300
Weight of objective function loss	50000
Weight of reconstruction loss	3
Termination tolerance on first order optimality of LBFGS optimizer	4×10^{-7}
Termination tolerance on parameter changes LBFGS optimizer	4×10^{-7}

H.4 BC

The Behavior Cloning (BC) algorithm [49] is a fundamental imitation learning method. BC aims to learn policies directly from expert demonstrations, leveraging supervised learning to map states to actions. This approach bypasses the need for exploration typically required in reinforcement learning by directly mimicking the behavior observed in the provided demonstrations. BC does not require interaction with the environment during training, which simplifies the learning process and reduces the computational resources needed.

In Behavior Cloning, the policy network is trained using supervised learning techniques, where the objective is to minimize the difference between the predicted actions and the actions taken by the expert in the training data. The loss function is typically the mean squared error between the predicted and expert actions. The training data, consisting of state-action pairs, is collected from expert demonstrations and stored in a dataset. In the inference, we evaluated the same objective function as the SAC, which is the MSE between the final state after $T-1$ steps' control and the target. For the partially observed or partially controlled settings, we concatenate zeros in the masked dimensions to make the same dimension as the input of the policy network, but the zeros are not considered during the calculation of the metrics. The detailed values of hyperparameters are provided in Table 13.

Table 13: **Hyperparameters of 1D BC.** The full observation partial control, partial observation full control, and partial observation partial control settings share the same hyperparameters.

Hyperparameter name	Value
Hyperparameters for 1D Burgers' equation control:	
Learning rate	1×10^{-4}
Training batch size	512
Number of episodes	5×10^5
Size of replay buffer	2×10^6
Number of layers of policy networks	2
Number of hidden dimensions of policy networks	1024
Activation function	ReLU

https://doi.org/10.52202/079017-0134

H.5 BPPO

The Behavior Proximal Policy Optimization (BPPO) algorithm [73] is an advanced reinforcement learning method that builds upon the strengths of Proximal Policy Optimization (PPO) while incorporating elements from behavior cloning. BPPO is an offline algorithm that monotonically improves behavior policy in the manner of PPO. Owing to the inherent conservatism of PPO, BPPO restricts the ratio of learned policy and behavior policy within a certain range, similar to the offline RL methods which make the learned policy close to the behavior policy. By leveraging the inherent conservatism of online on-policy algorithms, BPPO addresses the overestimation issue commonly encountered in offline RL settings. The algorithm starts by estimating a behavior policy using behavior cloning and then iteratively improves a target policy using the Proximal Policy Optimization (PPO) objective with a behavior constraint. Through this process of policy improvement, advantage estimation, and policy update, BPPO aims to refine the target policy while ensuring it remains close to the behavior policy. By combining the strengths of online on-policy methods with tailored offline RL techniques, BPPO demonstrates promising results on the D4RL benchmark, outperforming state-of-the-art offline RL algorithms.

During the training, BPPO first initializes the behavior policy π_β and target policy π_θ. Then, it estimates the behavior policy π_β using behavior cloning to mimic the behavior demonstrated in the offline dataset. Next, the target policy π_θ is optimized by the PPO objective with a behavior constraint, ensuring that the target policy remains close to the behavior policy. After that, it estimates the advantage function A^{π_β} using the behavior policy π_β to evaluate the quality of actions taken by the target policy. Finally, updating the target policy by maximizing the PPO objective with that estimated advantage function, and adjusting the policy parameters to improve performance. In the implementation, a state value network and Q value network are pre-trained using the state, action, and reward from the offline dataset.

In practice, to help the system approximate the target state accurately and quickly, we need to include the distance between the states of every time step and the target state in the reward. So the reward function of time step t, state u_t, target state u_d and action $\mathbf{w}_t$ here is defined as

$$r(t, \mathbf{u}_t, \mathbf{u}_d, \mathbf{w}_t) = -\int_\Omega |\mathbf{u}_t - \mathbf{u}_d|^2 \mathrm{d}\mathbf{x} - \alpha \int_\Omega |\mathbf{w}_t|^2 \mathrm{d}\mathbf{x},$$

where Ω is the space domain and α is the weight of energy. We take the Adam optimizer [29] to train the networks and update the temperature parameter. The detailed values of hyperparameters are provided in Table 14.

I 2D Jellyfish and Smoke Control Baselines

I.1 MPC and SL

Model Predictive Control (MPC) [57] is a control strategy that solves an optimization problem repeatedly to determine the optimal control inputs for a dynamic system. It operates over a finite prediction horizon, optimizing a cost function and applying only the first control action. In the 2D jellyfish movement control problem, MPC uses the control sequences $\mathbf{w}$ and the fluid states as internal state variables. Without the need to train a control agent model, MPC relies on 2D surrogate models mentioned in Appendix J to estimate future states based on the current state and control. We use backpropagation to compute the gradient, update the control action sequences, and optimize the control objective $\mathcal{J}$ in Eq. (19). For every time step, we get the optimized sequences from this time step forward in this way, and only the first control sequence of the optimized control sequences is applied. Compared with MPC, the Supervised learning (SL) method [24] only optimizes the entire control sequences and employs the entire sequences.

MPC is an optimization technique that aims to optimize the control of complex dynamic systems by considering future predictions. This approach can optimize performance measures over a future time horizon, handle systems with multiple variables and constraints, adapt to changes in the system behavior, and offer good performance even in the presence of nonlinearity. Nevertheless, using MPC comes with a high cost, both in terms of computational resources and time. Additionally, adapting the optimization hyperparameters for MPC can be a challenging task. In our experiment, both MPC and SL face difficulties when trying to generate smooth opening angle control curves, even

Table 14: **Hyperparameters of 1D BPPO.** The full observation partial control, partial observation full control, and partial observation partial control settings share the same hyperparameters.

Hyperparameter name	Value
Hyperparameters for 1D Burgers' equation control:	
State value network:	
Learning rate of value network	1×10^{-4}
Steps of value network	2×10^{6}
Number of layers of value network	3
Batch size of value network	512
Number of hidden dimensions of value network	512
Q value network:	
Learning rate of Q network	1×10^{-4}
Steps of Q network	2×10^{6}
Number of layers of Q network	2
Batch size of Q network	512
Number of hidden dimensions of Q network	1024
Target Q network updates per step	2
Soft update factor	0.005
Discount factor for reward	0.99
Behavior cloning:	
Learning rate of BC	1×10^{-4}
Training batch size of BC	512
Number of episodes of BC	5×10^{5}
BPPO:	
Number of episodes of BPPO	1×10^{2}
Number of layers of policy networks	2
Number of hidden dimensions of policy networks	1024
Learning rate of BPPO	1×10^{-5}
Training batch size of BPPO	512
Clip ratio of BPPO	0.25
Weight decay factor	0.96
Weight of advantage function	0.9
Size of replay buffer	2×10^{6}
Activation function	ReLU

when constraints $R(\hat{\mathbf{w}})$ are included in the optimization objective $\mathcal{J}$. In the case of multi-objective optimization problems, it becomes even more challenging for them to simultaneously achieve both the higher speed (bigger $\bar{v}$) and the control curves smoothness (smaller $R(\hat{\mathbf{w}})$).

I.2 SAC

For the 2D case, the algorithm and basic architecture of SAC are the same as the 1D case in Appendix H.2. When designing the reward function for the 2D jellyfish movement control, we find the periodic condition of the opening angle curves is hard to constrain. So to satisfy the periodic condition better, we include the distance between $\mathbf{w}_t$ of every time step t and $\mathbf{w}_0$. Also, we both consider the squared and absolute error of $(\mathbf{w}_t - \mathbf{w}_0)$ since they respectively constrain the periodic condition when $(\mathbf{w}_t - \mathbf{w}_0)$ is large and small. As a result, the reward function of time step t, force F_t, opening angle $(\mathbf{w}_{t-1}, \mathbf{w}_t)$ and condition angle $\mathbf{w}_0$ is defined as

$$r(t, \mathbf{w}_{t-1}, \mathbf{w}_t, \mathbf{w}_0) = (T - t) * F_t - \lambda_1(\mathbf{w}_t - \mathbf{w}_{t-1})^2 - \lambda_2((\mathbf{w}_t - \mathbf{w}_0)^2 + |\mathbf{w}_t - \mathbf{w}_0|),$$

where λ_1, λ_2 are weights of different terms. As for the 2D incompressible fluid control, we set the reward function as the percentage of smoke passing through the target bucket.

We take the Adam optimizer [29] to update the weights of networks and the temperature parameter as in the 1D experiment. The hyperparameters are reported in Table 15. In particular, we take the best

https://doi.org/10.52202/079017-0134

checkpoint to evaluate the final performance, thus the actual number of training episodes for each setting ranges from about 100 to about 300.

Table 15: **Hyperparameters of 2D SAC.** The full observation and partial observation settings share the same hyperparameters.

Hyperparameter name	Value
Hyperparameters for 2D Jellyfish movement control:	
Weight of the constraint of periodic condition β	0.001
Discount factor for reward	0.5
Target smoothing coefficient	0.05
Learning rate of critic loss	0.0003
Learning rate of entropy loss	0.0003
Learning rate of policy loss	0.0003
Training batch size	2048
Number of episodes	350
Number of model updates per simulator step	20
Value target updates per step	15
Size of replay buffer	11400001
Number of trajectories interacted with the environment per step	5
Activation function	ELU
Clipping's range of policy network's standard deviation output	$\left[e^{-5}, e^{-2}\right]$

I.3 BC

For the 2D task, the algorithm and basic architecture of BC are the same as the 1D case in Appendix H.4. In Behavior Cloning, the policy network is trained using supervised learning techniques, where the objective is to minimize the difference between the predicted actions and the actions taken by the non-expert in the training data. The loss function is typically the mean squared error between the predicted actions and actions in training data. The training data, consisting of state-action pairs, is collected from non-expert random demonstrations and stored in a dataset. In the inference, we evaluated the same objective function as the SAC. The detailed values of hyperparameters are provided in Table 16.

Table 16: **Hyperparameters of 2D BC.** The full observation and partial observation settings share the same hyperparameters.

Hyperparameter name	Value
Hyperparameters for 2D control:	
Learning rate	1×10^{-4}
Training batch size	512
Number of episodes	5×10^3
Size of replay buffer	2×10^6
Activation function	ELU
Clipping's range of policy network's standard deviation output	$[-5, -2]$

I.4 BPPO

For the 2D task, the algorithm and basic architecture of BPPO are the same as the 1D case in Appendix H.5. In jellyfish movement control, when designing the reward function, we find the periodic condition of the opening angle curves is hard to constrain. So to satisfy the periodic condition better, we include the distance between $\mathbf{w}_t$ of every time step t and $\mathbf{w}_0$. Also, we both consider the squared and absolute error of $(\mathbf{w}_t - \mathbf{w}_0)$ since they respectively constrain the periodic condition when $(\mathbf{w}_t - \mathbf{w}_0)$ is large and small. As a result, the reward function of time step t, force

F_t, opening angle $(\mathbf{w}_{t-1}, \mathbf{w}_t)$ and condition angle $\mathbf{w}_0$ is defined as

$$r(t, \mathbf{w}_{t-1}, \mathbf{w}_t, \mathbf{w}_0) = (T - t) * F_t - \lambda_1(\mathbf{w}_t - \mathbf{w}_{t-1})^2 - \lambda_2((\mathbf{w}_t - \mathbf{w}_0)^2 + |\mathbf{w}_t - \mathbf{w}_0|),$$

where λ_1, λ_2 are weights of different terms. The hyperparameters are reported in Table 17.

Table 17: **Hyperparameters of 2D BPPO.** The full observation and partial observation settings share the same hyperparameters.

Hyperparameter name	Value
Hyperparameters for 2D control:	
State value network:	
Learning rate of value network	1×10^{-4}
Steps of value network	2×10^3
Number of layers of value network	3
Batch size of value network	512
Q value network:	
Learning rate of Q network	1×10^{-4}
Steps of Q network	2×10^3
Number of layers of Q network	3
Batch size of Q network	512
Target Q network updates per step	2
Soft update factor	0.005
Discount factor for reward	0.99
Behavior cloning:	
Learning rate of BC	1×10^{-4}
Training batch size of BC	512
Number of episodes of BC	5×10^3
BPPO:	
Number of episodes of BPPO	1×10^2
Number of layers of policy networks	3
Learning rate of BPPO	1×10^{-5}
Training batch size of BPPO	512
Clip ratio of BPPO	0.25
Weight decay factor	0.96
Weight of advantage function	0.9
Size of replay buffer	2×10^6
Activation function	ELU

J Surrogate Models

J.1 1D Surrogate Model

For the control problem of 1D Burgers' equation, our 1D surrogate model is based on the previous paper [24], which uses 2 autoencoders to model dynamics in the latent space. The neural simulator architecture and training details are shown in Table 18.

J.2 2D Jellyfish Force Models

Dataset. In 2D jellyfish movement control experiments, we train a force surrogate to approximate the computation of the average speed of the jellyfish for the guidance of inference, which is implemented by a neural network and is thus differentiable. The training data consists of pressure, boundary, and force data in the training trajectories. Each training trajectory amounts to $\tilde{T} = 40$ training samples. Therefore, we have 1.2 million training samples and 4 thousand testing samples in total.

Model. The model's input contains pressure, boundary mask, and offsets with shape $4 \times 64 \times 64$ at a certain time step, and the output is the corresponding forces of x and y directions. The model

 https://doi.org/10.52202/079017-0134

Table 18: **Hyperparameters of 1D surrogate model**.

Hyperparameter name	Full observation, partial control	Partial observation, full control	Partial observation, partial control
Autoencoder of state			
Convolution kernel size	5	5	5
Convolution padding	2	2	2
Activation function	ELU	ELU	ELU
Latent vector size	256	128	128
Autoencoder of force			
Convolution kernel size	5	5	5
Convolution padding	2	2	2
Activation function	ELU	ELU	ELU
Latent vector size	256	256	256
Training			
Training batch size	5100	5100	5100
Optimizer	Adam	Adam	Adam
Learning rate	1e-3	1e-3	1e-3
Training epochs	500	500	500
Learning rate scheduler	cosine annealing	cosine annealing	cosine annealing

architecture is the down-sampling part of a U-Net [55] that embeds the input features into a 512-dimensional hidden representation; then we use a linear function with output dimension two to output forces.

Training. We use MSE (mean squared error) loss between the ground truth and predicted forces to train the force surrogate model. The optimizer is Adam [29]. The batch size is 64. The model is trained for 10 epochs. The learning rate starts from 1×10^{-4} and multiplies a factor of 0.1 every three epochs. After training, the relative l_2 test error is 0.4%.

J.3 2D Jellyfish Boundary Mask and Offsets Updater

Dataset. In 2D jellyfish movement control experiments, we train a boundary mask and offsets updater surrogate to approximate the transition of boundary mask and offsets from time step 0 to t. Thus each training trajectory amounts to $\tilde{T} - 1 = 39$ training samples.

Model. The input is the boundary mask and offsets at time step 0 with shape $3 \times 64 \times 64$, and the difference of the opening angle from 0 to t. The output is the boundary mask and offsets at time step t with shape $3 \times 64 \times 64$. The model architecture is the U-Net [55] with additional scalar input, similar to the denoising network in DDPM [19], where the input scalar diffusion step is replaced by the angle difference in our model.

Training. We use MSE (mean squared error) loss between the ground truth and predicted boundary mask and offsets to train this surrogate model. Hyperparameters of training are the same as those of the force surrogate model.

J.4 2D Jellyfish Simulator

Dataset. In 2D jellyfish movement control experiments, we need to train a surrogate model as a solver of the physical dynamics for the baseline methods like SAC (online) and MPC, because of their iterative nature. Conversely, our diffusion method *does not* need this surrogate model. This model approximates the transition of states under the boundary condition from time step t to $t + 1$. Thus each training trajectory amounts to $\tilde{T} - 1 = 39$ training samples. We train two versions of this model for full/partial observation settings.

Model. This surrogate model is also implemented by the U-Net [55]. The model input is the states, boundary mask, and offsets at time t with shape $6 \times 64 \times 64$ for the full observation setting and $4 \times 64 \times 64$ for the partial observation setting. The output is the predicted states at time step $t + 1$, with shape $3 \times 64 \times 64$ for the full observation setting and $1 \times 64 \times 64$ for the partial observation setting.

Table 19: **More results of 1D Burgers's equation control.** *Italic* font denotes unfair results, bold font denotes the best model among fairly evaluated baselines, and underline denotes the second best model among them.

	PO-FC $\mathcal{J}\downarrow$	FO-PC $\mathcal{J}\downarrow$	PO-PC $\mathcal{J}\downarrow$
SAC (online)	0.01567	0.034092	0.02768
DiffPhyCon-lite	<u>0.01139</u>	**0.00037**	**0.00494**
DiffPhyCon	**0.01103**	**0.00037**	**0.00494**

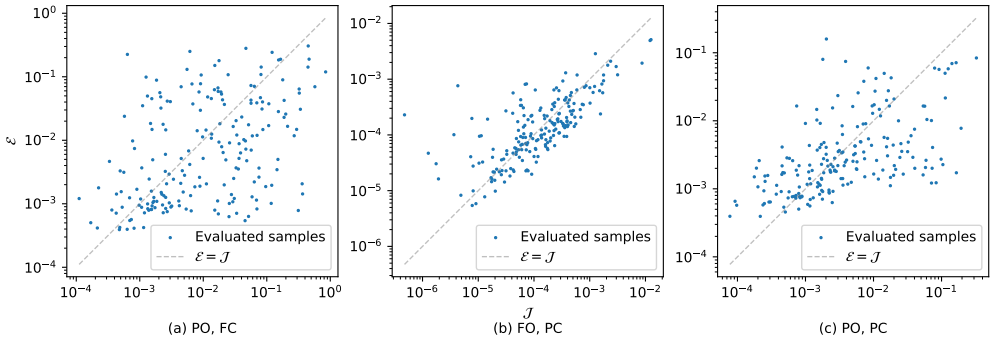


Figure 16: $\mathcal{J}$ **and state MSE** $\mathcal{E}$ **of different samples in different settings.**

Training. We use MSE loss between the ground truth and predicted states to train this surrogate model. Hyperparameters of training are the same as those of the force surrogate model.

K Additional Results of Experiments

K.1 DiffPhyCon for 1D Burgers' Equation

Since online RL methods are effective in a wide domain of control tasks, we tested the performance of the classical RL method, SAC (online), as shown in Table 19. The online SAC belongs to a different setting than the offline training setting considered in our work. Even with access to more information from the environment, online SAC is less competitive to our DiffPhyCon in Burgers' control task.

As DiffPhyCon can jointly generate the trajectory $\mathbf{u}$ and control sequence $\mathbf{w}$, we demonstrate the MSE of the diffused trajectory $\mathbf{u}$ from the trajectory computed by the ground-truth solver $\mathbf{u}_{\text{ground truth}}$ given the diffused control sequence $\mathbf{w}$. This should be differentiated from $\mathcal{J}$ reported in Table 1 which is the MSE of the trajectory computed by the ground-truth solver given $\mathbf{w}$ from the target trajectory $\mathbf{u}^*$. The former MSE is denoted as $\mathcal{E} := \|\mathbf{u} - \mathbf{u}_{\text{ground truth}}\|$ in Figure 16.

It can be observed from Figure 16 that there is a clear correlation between $\mathcal{J}$ and $\mathcal{E}$. This is because, the higher the prediction error is, the less accurate the solution to the inverse control problem would be. The correlation proves that our DiffPhyCon learns the dynamics of the system (i.e., the dependency of $\mathbf{u}$ on $\mathbf{w}$), based on which the control sequence is generated. In partial observation settings, the randomness is larger than in the full observation setting, which demonstrates the challenge of controlling partially observed systems. The reference line denotes $\mathcal{J} = \mathcal{E}$ where the control objective completely depends on the prediction accuracy. In Figure 16 (b), samples cluster tightly around the line, which implies DiffPhyCon can find near-optimal control sequence based on the predicted trajectory in the corresponding level of prediction accuracy. This ability is remarkable in that DiffPhyCon does not rely on explicit gradient descent like those in the adjoint method and SL.

K.2 2D Jellyfish Movement

In this subsection, we record extensive results of 2D jellyfish movement control. To further compare baselines with DiffPhyCon in more settings, we change ζ in $\mathcal{J}$ and results of $\zeta = 500$, $\zeta = 1000$ and

https://doi.org/10.52202/079017-0134

Table 20: **Full observation setting of 2D jellyfish movement control.** Bold font denotes the best model, and underline denotes the second best model.

	$\zeta = 500$			$\zeta = 1000$			$\zeta = 2000$		
	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$
MPC	-38.21	0.1743	212.49	25.72	0.0112	109.17	44.24	0.0755	31.29
SL	90.09	0.2570	166.89	-76.94	0.1286	205.57	-62.96	0.0648	127.8
SAC (pseudo-online)	-134.13	0.0204	154.49	-166.96	0.0069	178.14	-309.18	0.0040	313.19
SAC (offline)	-159.83	0.0119	171.73	-158.66	0.0069	165.58	-278.52	0.0051	283.58
BC	15.61	0.0589	43.33	30.48	0.0629	32.44	33.5	0.0677	34.23
BPPO	88.64	0.0776	-11.07	<u>107.67</u>	0.0867	<u>-20.93</u>	-20.12	0.0621	82.27
DiffPhyCon-lite	<u>151.57</u>	0.0939	<u>-57.69</u>	95.04	0.0746	-20.47	<u>150.53</u>	0.0923	<u>-58.23</u>
DiffPhyCon	**310.02**	0.2344	**-75.64**	**279.87**	0.2058	**-74.11**	**270.56**	0.1755	**-95.08**

Table 21: **Partial observation setting of 2D jellyfish movement control.** Bold font denotes the best model, and underline denotes the second best model.

	$\zeta = 500$			$\zeta = 1000$			$\zeta = 2000$		
	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$	$\bar{v} \uparrow$	$R(\mathbf{w}) \downarrow$	$\mathcal{J} \downarrow$
MPC	-179.85	0.2534	433.3	-150.51	0.1791	329.59	-116.75	0.1397	256.46
SL	44.04	0.2895	245.43	-102.98	0.1188	221.79	-152.45	0.0661	218.53
SAC (pseudo-online)	21.87	0.0798	57.96	-153.09	0.0057	158.82	-206.43	0.0048	211.2
SAC (offline)	-149.32	0.0156	164.93	-206.21	0.0058	211.96	-258.96	0.0029	261.87
BC	35.49	0.057	21.48	20.08	0.0556	35.48	<u>35.14</u>	0.0594	<u>24.27</u>
BPPO	<u>104.28</u>	0.0746	<u>-29.67</u>	<u>54.83</u>	0.0518	<u>-3.02</u>	16.97	0.0598	42.83
DiffPhyCon-lite	-0.35	0.0782	78.56	2.92	0.0779	74.97	-10.89	0.0760	86.93
DiffPhyCon	**173.27**	0.1636	**-9.66**	**150.21**	0.1269	**-23.32**	**101.25**	0.1025	**1.21**

$\zeta = 2000$ are listed in Table 20 and Table 21. The results show that DiffPhyCon outperforms other baselines in diverse settings with different objectives.

Table 22: **Results of myopic failure modes of SAC on 2D jellyfish movement control**.

λ_0	Average speed ($\bar{v}$)	$R(\mathbf{w})$	objective $\mathcal{J}$	periodicity error
SAC (weight=100)	59.82	0.0263	-33.48	0.35187
SAC (weight=1000)	-166.96	0.0112	178.14	0.02299
DiffPhyCon	279.87	0.2058	-74.11	0.01157

K.3 Myopic failure modes of SAC

To further confirm the advantage of diffusion models over the baseline SAC in alleviating the myopic failure modes, we perform additional 2D experiments. Specifically, when solving the jellyfish movement problem using SAC, we incorporated a constraint to achieve periodic motion of the jellyfish in the reward function, namely, we added a hyperparameter as the weight of the term $d(w_T, w_0)$ in Eq. (19). The challenge caused by the periodicity condition is that the average speed and $R(\mathbf{w})$ are calculated over the whole horizon while the periodicity condition is only evaluated at the final time step of the horizon. Thus, when the control objective consists of all three terms, it requires global optimization over the whole horizon to balance different terms. In Table 22, we present the original results (weight=1000, corresponding to Table 3) and the results after reducing the weighting of this term (weight=100). It can be observed that while the speed of the jellyfish increases (though still not reaching the results of DiffPhyCon), the $R(\mathbf{w})$ and constraint term for periodicity sharply rise. From these results and the additional results in Table 20 and Table 21 where we test the performance of SAC under different weights ζ of $R(\mathbf{w})$, we conclude that SAC struggles to provide satisfactory policies that simultaneously satisfy multiple conflicting constraints with different time coverage, resulting in a myopic failure model. The reason may be that it is difficult to estimate the effect of each term in the control objective in each time step in the iterative planning fashion [1], which this issue could be well addressed by our diffusion models with an inherent nature of global optimization.

Table 23: **Efficiency comparison on 1D Burgers' equation.** Inference time is tested on a Tesla-V100 GPU with 8 CPUs.

Methods	Training time (hours)	Inference time (seconds)
DiffPhyCon-lite	1.7 (1 A100-80G GPU, 8 CPUs)	21.13
DiffPhyCon	4.4 (1 A100-80G GPU, 8 CPUs)	58.97
DiffPhyCon-DDIM (8 sampling steps)	1.7 (1 A100-80G GPU, 8 CPUs)	0.53
PID	0.1 (1 A100-80G GPU, 8 CPUs)	3.61
SL	2.6 (1 V100-32G GPU, 12 CPUs)	74.85
SAC	10.5 (1 A6000-48G GPU, 16 CPUs)	0.11
BC	8.8 (1 V100-32G GPU, 12 CPUs)	1.22
BPPO	8.9 (1 V100-32G GPU, 12 CPUs)	0.82

Table 24: **Efficiency comparison on 2D jellyfish movement.** Inference time is tested on a Tesla-V100 GPU with 8 CPUs.

Methods	Training time (hours)	Inference time (seconds)
DiffPhyCon	62 (2 A100-80G GPUs, 32 CPUs)	252.2
DiffPhyCon-DDIM (8 sampling steps)	62 (2 A100-80G GPUs, 32 CPUs)	12.6
MPC	52.1 (1 A100-80G GPU, 16 CPUs)	1401.7
SL	52.1 (1 A100-80G GPU, 16 CPUs)	133.5
SAC	9.5 (1 A100-80G, 16 CPUs)	0.2
BC	2.8 (1 A100-80G, 16 CPUs)	0.99
BPPO	3.0 (1 A100-80G GPU, 16 CPUs)	1.08

K.4 Efficiency Comparison

We test the training and inference (control) efficiency of DiffPhyCon and baselines on both 1D Burgers' equation and 2D jellyfish movement tasks. On the 1D Burgers' equation task, we test the efficiency of partial observation and full control (PO-FC); on the 2D jellyfish movement task, we test the efficiency of full observation (FO).

The results are presented in Table 23 and Table 24. For training efficiency, since models with different sizes are trained on different machines, we report the training time of models along with the machine information. The inference efficiency is tested on a single Tesla-V100 GPU with 8 CPUs and the average inference times over all test samples are reported. For training efficiency, we have the following observations. On 1D Burgers' equation, our DiffPhyCon is comparable with SL and more efficient than reinforcement learning (RL) methods like SAC, BC, and BPPO. On 2D jellyfish movement, our DiffPhyCon costs similar training time compared with MPC and SAC, but more time than RL methods. For inference efficiency, these results reveal that although DiffPhyCon is not as efficient as RL methods, it has competitive efficiency compared to SL and MPC. In particular, by using the fast sampling method DDIM [60], DiffPhyCon could be accelerated significantly by reducing the number of sampling steps.

L Effect of Hyperparameters

L.1 Effect of γ

1D Burgers' Equation control. In the 1D Burgers control task, the prior reweighting is intrinsically unnecessary. The evaluated control target follows the same distribution as the training set, and the samples in the training set are defined to be the optimal control. Therefore, the generated control sequences following $p(w|u_0, u_T)$ are already the (near) optimal distribution which alleviates the need for prior reweighting.

We tested different scheduling of the prior reweighting γ and found their performance similar. Finally, we use the same cosine schedule as that in the DDPM noise for clarity. Our experiment results also revealed that the performance of DiffPhyCon in 1D Burgers equation control is insensitive to the prior reweighting intensity γ as we reported in Table 1. Besides, we conducted a more comprehensive experiment on the prior reweighting intensity γ in all three settings (FO-PC; PO-FC; PO-PC) as

https://doi.org/10.52202/079017-0134

Table 25: **Results of different γ in DiffPhyCon on FO-PC 1D Burgers equation control task.**

γ	0.0	0.3	0.5	0.7	1.0
$\mathcal{J}_{\text{actual}}$	0.00038	0.00037	0.00037	0.00037	0.00037
State MSE	0.00063	0.00043	0.00034	0.00028	0.00025

Table 26: **Results of different γ in DiffPhyCon on PO-FC 1D Burgers equation control task.**

γ	0.0	0.3	0.5	0.7	1.0
$\mathcal{J}_{\text{actual}}$	0.01159	0.01155	0.01152	0.01148	0.01139
State MSE	0.02643	0.02642	0.02642	0.02641	0.02639

in Table 25, 26, and 27. The results show that prior reweighting does not significantly impact the performance of DiffPhyCon in 1D Burgers control task, where State MSE denotes the deviation between generated u and the ground truth u_{gt} given by the numerical simulator based on generated f.

2D jellyfish movement control. Performance of DiffPhyCon is determined by the hyperparameter γ. Since diffusion models denoise gradually, we use a varying sequence of $\gamma = \{\gamma_k\}_{k=1}^K$ to subtract prior in DiffPhyCon. Specifically, the schedule of γ is set as $\gamma_k = 1 - \xi \cdot \beta_{K-k}, k = 1, \cdots, K$, where ξ is a fixed coefficient to control the scale of γ and $\beta = \{\beta_k\}_{k=0}^K$ is the schedule of variances of noise in DDPM [19]. In our implementation, we use the sigmoid schedule of β [25]. The total number of inference steps is $K = 1000$. Thus we only need to tune ξ to examine the effect of γ. When $\xi < 0$, DiffPhyCon is prone to restrict $\mathbf{w}$ within its prior distribution of training dataset in inference. When $\xi > 0$, DiffPhyCon is more likely to generate new kinds of $\mathbf{w}$ beyond training ones. When $\xi = 0$, DiffPhyCon degenerates to DiffPhyCon-lite. In 2D experiments, We set default $\xi = 0.3$ and the corresponding $\gamma_1 = 0.7$ as we empirically find this value performs well and steadily. We present the performance of DiffPhyCon on the 2D jellyfish movement control task under different γ in Figure 17 and Table 28. We can observe that the performance increases along with decreasing of γ_1. When $\gamma_1 < 0.6$, invalid generated control sequences emerge because the prior is largely overlooked. Thus the valid interval for γ of prior reweighting on this task is $[0.6, 1.0]$. It is interesting to find that when $\gamma_1 > 1$, the performance decreases. This may be caused by the strict constraint of the prior distribution of $p(\mathbf{w}, \mathbf{c})$, which results in generating control sequences similar to those from training datasets and thus not good.

L.2 Effect of λ

1D Burgers' Equation control. In 1D Burgers control, DiffPhyCon uses guidance conditioning since it empirically performs better than explicit guidance as shown in Table 29. Therefore, our model does not use λ to generate the optimal control. However, when we consider the energy cost as in Figure 3, we use explicit guidance to control the magnitude of control signals as it is a more natural way to incorporate the objective $\lambda\mathcal{J}_{\text{energy}}$ into the overall diffusion objective $E_\theta^{(\gamma)}(u, w, c) + \lambda\mathcal{J}_{\text{energy}}$ where the condition c is $u_T = u_d$.

The guidance intensity uses a cosine schedule [43] where $\lambda = \lambda_0\beta_k,\ k = 1, 2, ..., K$ where β_k starts from $\beta_1 = 0.001$ to $\beta_K = 1$. The schedule decreases as the diffusion step decreases during inference, where the gradually decreasing guidance intensity provides strong guides in the initial stage while reducing the impact on the near-clean samples.

Table 27: **Results of different γ in DiffPhyCon on PO-PC 1D Burgers equation control task.**

γ	0.0	0.3	0.5	0.7	1.0
$\mathcal{J}_{\text{actual}}$	0.00493	0.00493	0.00493	0.00493	0.00494
State MSE	0.00675	0.00667	0.00665	0.00664	0.00663

Table 28: **Results of different γ in DiffPhyCon on 2D jellyfish movement control task**.

γ_1	ξ	Average speed ($\bar{v}$)	$R(\mathbf{w})$	objective $\mathcal{J}$
0.6	0.4	410.6	0.2581	-152.51
0.7	0.3	279.87	0.2058	-74.11
0.8	0.2	197.18	0.1312	-65.99
0.9	0.1	76.97	0.0741	-2.84
1.0	0	95.04	0.0746	-20.47
1.1	-0.1	81.41	0.0742	-7.21
1.2	-0.2	84.56	0.0736	-10.93
1.3	-0.3	65.12	0.0725	7.38
1.4	-0.4	65.02	0.0734	8.43
1.5	-0.5	64.07	0.0752	11.1

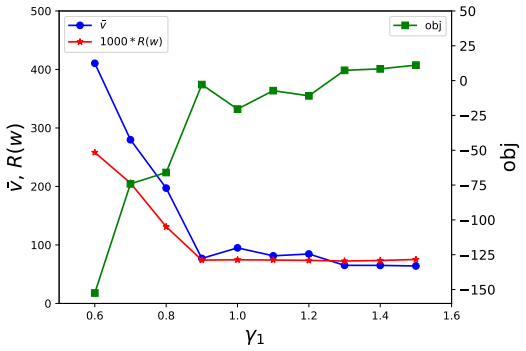


Figure 17: **Results of different γ in DiffPhyCon on 2D jellyfish movement control task**.

In such a setting, varying λ changes how the generated samples are shifted towards lower energy regions, but an excessively large λ causes the sample to deviate from the physically feasible region. Results in Table 30, 31, and 32 show that the behavior of DiffPhyCon under different λ is expected. Note that "physically feasible" samples here are samples with lower $\mathcal{J}_{\text{actual}}$.

Table 29: **Results of two different guidance type of DiffPhyCon in 1D Burgers equation control task.**

Guidance Type	FO-PC	PO-FC	PO-PC
Explicit Guidance	0.02789	0.03257	0.05584
Conditioning Guidance	0.00037	0.01139	0.00494

2D jellyfish movement control. In our jellyfish movement control task, we use explicit guidance with λ involved. Since diffusion models denoise gradually, similar to the hyperparameter γ, we use a varying sequence of $\lambda = \lambda_k = \lambda_0 \times \beta_k$, where β_k is the schedule of variances of noise in DDPM [2] increasing monotonically from $\beta_1 = 0.0003$ to $\beta_K = 1$. The motivation of this schedule is that the confidence of gradient estimation becomes stronger along with the denoising process from $k = K$ to $k = 1$. The results are shown in the Table 33. We can observe that as λ_0 (which determines the λ schedule) increases from 0 to 0.6, the performance of DiffPhyCon improves consistently. Compared to Table 3, DiffPhyCon always outperforms baselines for a wide range of $0.1 \leq \lambda_0 \leq 0.5$. When $\lambda_0 \geq 0.6$, the results are no longer valid since the generated control sequences may be unfeasible. This reflects that λ should not be overly large to make it safe, which is a consistent conclusion with our previous analysis.

https://doi.org/10.52202/079017-0134

Table 30: **Results of different λ for $\mathcal{J}_{\text{energy}}$ of DiffPhyCon in FO-PC 1D Burgers equation control task.**

λ	0	10	100	500	1000	10000	100000	1000000
$\mathcal{J}_{\text{actual}}$	0.00037	0.00037	0.00037	0.00036	0.00037	0.00147	0.02202	0.06444
State MSE	0.00025	0.00025	0.00024	0.00024	0.00024	0.00083	0.00802	0.01838
$\mathcal{J}_{\text{energy}}$	1320.98206	1309.31104	1237.06958	1061.67761	946.07617	584.32098	225.50281	38.25005

Table 31: **Results of different λ for $\mathcal{J}_{\text{energy}}$ of DiffPhyCon in PO-FC 1D Burgers equation control task.**

λ	0	10	100	500	1000	10000	100000	1000000
$\mathcal{J}_{\text{actual}}$	0.01139	0.01152	0.01252	0.01906	0.02100	0.04411	0.08472	0.11934
State MSE	0.02639	0.02641	0.02661	0.02919	0.02990	0.03857	0.05472	0.06471
$\mathcal{J}_{\text{energy}}$	862.98633	860.05164	839.11932	770.45819	704.68665	311.21915	69.87619	8.93373

M Extensions to Finer-grained Jellyfish Movement Control Task

To test the scalability of DiffConPDE in handling high-dimensional and complex dynamic systems, we extend the jellyfish movement control experiment from rigid boundaries of wings to soft ones.

M.1 Experimental Setting

Similar to previous setups in Appendix F, this extension aims to control the movement of a flapping jellyfish with two wings in a 2D fluid field where fluid flows with constant initial speed. However, in this task, we parametrize the jellyfish's boundary as the coordinate change (Δx, Δy) for each cell within the boundary, which serves as the control sequence $\mathbf{w}$. Undoubtedly, this is a high-dimensional and complex control task. Firstly, the control sequence $\mathbf{w}$ is elevated to three dimensions. In addition to ensuring optimization towards the control objective, maintaining consistency in the movement of different cells within the jellyfish boundary is also crucial. Bearing this in mind, we have chosen to employ DiffConPDE-lite for this experimental setup to minimize potential disruptions to the jellyfish boundary. Regarding the PDE state, the experiment adopts the full observation setup. All other settings remain the same with Appendix F.

M.2 Data Generation

In this extension, we leverage data previously generated under the vanilla setting. Additionally, we utilize opening angles Θ, The coordinates of the rotation center $\mathbf{h}$, and the boundary mask to generate the required control sequence $\mathbf{w}$. We represent $\mathbf{w}$ as a tensor of shape $[\tilde{T}, 2, 64, 64]$. Each cell has two features representing the changes in the two coordinates. For each trajectory, $\mathbf{w}$ is generated as follows: Firstly, for the initial $\mathbf{w}_0$ and final $\mathbf{w}_T$, we set all elements in the tensor to 0. Then, for each $w_{[1,T-1]}$ corresponding to the cells within the boundary, we first obtain the relative coordinates $c = (x, y)$ of the cell with respect to the rotation center using the boundary mask and the coordinates of the rotation center $\mathbf{h}$. Afterwards, we construct the rotation matrix $M_t(\Theta_t)$ at different time steps using the opening angle $\Theta_{[1,T-1]}$:

$$M_t(\Theta_t) = \begin{bmatrix} \cos(\Theta_t) & -\sin(\Theta_t) \\ \sin(\Theta_t) & \cos(\Theta_t) \end{bmatrix}.$$

The coordinate change w_t can be obtained using the following formula:

$$w_t = cM_t - c$$

Table 32: **Results of different λ for $\mathcal{J}_{\text{energy}}$ of DiffPhyCon in PO-PC 1D Burgers equation control task.**

λ	0	10	100	500	1000	10000	100000	1000000
$\mathcal{J}_{\text{actual}}$	0.00494	0.00500	0.00549	0.00794	0.01426	0.02411	0.03979	0.08153
State MSE	0.00663	0.00676	0.00768	0.01048	0.01525	0.02766	0.03635	0.05459
$\mathcal{J}_{\text{energy}}$	1314.90698	1297.32849	1192.89026	945.77179	793.28442	331.45758	108.39325	16.13734

Table 33: **Results of different λ on 2D jellyfish movement control**.

λ_0	Average speed ($\bar{v}$)	$R(\mathbf{w})$	objective $\mathcal{J}$
0.6	-	-	-
0.5	362.82	0.192	-170.79
0.4	322.04	0.182	-140.04
0.3	279.87	0.206	-74.11
0.2	192.77	0.114	-78.57
0.1	103.89	0.091	-12.32
0	-86.69	0.042	128.67

Additionally, for cells that are not within the boundary, the corresponding $w_{[1,T-1]}$ is also directly set to $(0, 0)$. Therefore, all the data involved in this experiment are as follows:

- Coordinate change of cells $\mathbf{w}$: shape $[\tilde{T}, 2, 64, 64]$. For each step, we save the coordinate change of each cell within the boundary.
- PDE states u: shape $[\tilde{T}, 3, 64, 64]$. For each step, we save the states of the fluid field consisting of velocity in x and y directions and pressure. To save space, we downsample the resolution from 128×128 to 64×64.
  - velocity: $[\tilde{T}, 2, 64, 64]$.
  - pressure: $[\tilde{T}, 1, 64, 64]$.
- opening angels Θ: shape $[\tilde{T}]$. For each step, we save the opening angle in radians.
- boundary mask and offsets b: $[\tilde{T}, 3, 64, 64]$. For each step, we save the mask of merged wings with half coordinates of boundary points and offsets in both $\mathbf{x}$ and y directions. The resolution is 64×64, compatible with that of the states.
  - mask: $[\tilde{T}, 1, 64, 64]$.
  - offsets: $[\tilde{T}, 2, 64, 64]$.
- force: shape $[\tilde{T}, 2]$. For each step, the simulator outputs the horizontal and vertical force from the fluid to the jellyfish. The horizontal force is regarded as a thrust to jellyfish if positive and a drag otherwise.

M.3 Model

Similar to the vanilla setting, this experiment also employs U-net as the backbone for the diffusion model. The U-net architecture remains consistent. The input of the U-net includes PDE state $\mathbf{u}([T, 3, 64, 64])$, control sequence $\mathbf{w}([T, 2, 64, 64])$, initial boundary mask and offset $([1, 3, 64, 64])$. It is worth noting that to align the initial mask and offset with others in terms of shape, we expand them along the time dimension, resulting in the final shapes of $[T, 3, 64, 64]$. Therefore, the shape of the input to the model is $[T, 8, 64, 64]$, and the output of the model includes both noise of the predicted state and the control sequence, with a shape of $[T, 5, 64, 64]$.

M.4 Training, Inference, and Evaluation

Training. During the training phase, the diffusion-flow task adopts a similar setup, utilizing the Mean Squared Error (MSE) between the model predictions and Gaussian noise as the loss function (Eq.4). The batch size is chosen as 16, and the training involves 180,000 iterations. The learning rate starts at 1×10^{-4} and increases by a factor of 0.1 at the 50,000th and 150,000th iterations. More training details are presented in Table 34.

Inference. The pipeline of inference is shown in Figure 18. Both diffused variables $\mathbf{u}_{[0,T]}$ and $\mathbf{w}_{[0,T]}$ are initialized from Gaussian prior and gradually denoised from denoising step $k = 1000$ to $k = 0$ based on denoising networks and guidance. Similar to the vanilla setting, due to the introduction of the initial boundary mask and offset as additional inputs, the number of channels in

https://doi.org/10.52202/079017-0134

Table 34: **Hyperparameters of network architecture and training for the Finer-grained Jellyfish Boundary Control Task**.

Hyperparameter name	full observation
Batch size	16
Optimizer	Adam
Learning rate	0.001
Loss function	MSE

the model's input and output are inconsistent. For guidance, we fix $\lambda = 5 \times 10^{-5}$ and we also utilized a surrogate model to approximate the force of fluid on the jellyfish. In the k-th denoising step, the inputs of this model include the noise-free state $\hat{\mathbf{u}}_{[0,T],k}$, $\hat{\mathbf{w}}_{[0,T],k}$ estimated from $\mathbf{u}_{[0,T],k}$,$\mathbf{w}_{[0,T],k}$ by Eq. (6), initial boundary mask and offset. The model output is the force. The remaining aspects such as model architecture, training details, etc., remain the same. As for the regularization term $R(\mathbf{w})$, we approximate the opening angle of the jellyfish's wings at time t using the change of the coordinate $(\Delta x_t, \Delta y_t)$ and the initial coordinates (x, y) of each cell to achieve regularization. Firstly, we utilize the following formula to approximate the angle change of each cell:

$$\hat{\theta}_t = \sqrt{(\Delta x_t^2 + \Delta y_t^2)/(x^2 + y^2)}$$

Then, we calculate the average value of the angles of different cells, which serves as an approximation of the angle between the two wings of the jellyfish at time t, denoted as $\hat{\Theta}_t$. It is evident that optimizing this term intuitively minimizes the coordinate changes of each cell, aligning with the objective of regularization. The control objective $\mathcal{J}$ in Eq. (19) is computed as a summation of force and $R(\hat{\mathbf{w}}_{[0,T]})$, and we fix $\zeta = 1000$. Then the gradients of the objective $\mathcal{J}$ in terms of $\hat{\mathbf{u}}_{[0,T]}$ and $\hat{\mathbf{w}}_{[0,T]}$ are computed and used in guidance. Because this experiment selects DiffConPDE-lite, all these gradients are directly subtracted from $[\mathbf{u}_{[0,T],k}, \mathbf{w}_{[0,T],k}]$ to generate $[\mathbf{u}_{[0,T],k-1}, \mathbf{w}_{[0,T],k-1}]$.

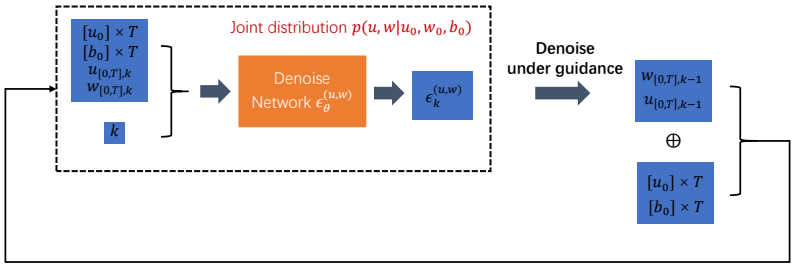


Figure 18: **Inference of our finer-grained jellyfish boundary control task**.

Evaluation The inference outputs coordinate change for each cell of $T = 20$ steps for 50 testing samples. To align with the evaluation metrics of the vanilla setting, we first need to calculate the opening angle Θ_t using the coordinate change of the cells at time t. Firstly, at time $t \in [1, T-1]$, for each cell, we obtain the rotated coordinates (x'_t, y'_t) using the initial coordinates (x, y) and the coordinate change $(\Delta x_t, \Delta y_t)$. Then, combined with the coordinates of the rotation center $\mathbf{h}$, we can respectively obtain the radial distance of the cell relative to the rotation center before and after rotation, denoted as r and r'. Then, according to the law of cosines, we can obtain the rotation angle $\delta\theta$ of the cell at time t compared to the initial time:

$$|\Delta\theta_t| = \frac{r_t'^2 + r_t^2 - (\delta x_t^2 + \delta y_t^2)}{2 r'_t r_t}.$$

Afterward, we take the average of the angle changes of all cells within the boundary as the angle change of the jellyfish's two wings at time t, denoted as $\Delta\Theta_t$. Finally, based on the average value of

the flow in the x-direction at time t, the opening or closing status of the jellyfish relative to the initial moment is determined. If it is negative, it indicates opening; otherwise, it suggests closing. This process allows us to obtain the specific angle change for each time step compared to the first moment:

$$\Theta_t = \begin{cases} \Theta_0 + |\Delta\Theta_t| & \Delta\bar{x}_t < 0 \\ \Theta_0 - |\Delta\Theta_t| & \Delta\bar{x}_t > 0 \\ \Theta_0 & \Delta\bar{x}_t = 0. \end{cases}$$

After obtaining the theta sequence, we can follow the vanilla evaluation procedure.

M.5 Results

We present the results in Table 35. From the table, it can be observed that our method optimized the objective $\mathcal{J}$ to 84.31 for this task. Compared to the results based on the original setting in Section 4.2, our method is still competitive with baselines (shown in Table 3) in this more challenging setting. An example of optimized soft boundary shapes is illustrated in Figure 19. These results demonstrate the good scalability of our method when facing higher-dimensional and complex control tasks.

Table 35: **Performance of DiffPhyCon-lite on the finer-grained jellyfish boundary control task**.

Average speed ($\bar{v}$)	$R(\mathbf{w})$	objective ($\mathcal{J}$)
-33.91	0.0504	84.31

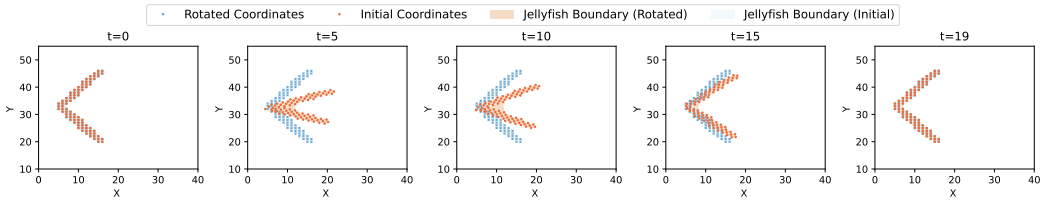


Figure 19: **Inference results of soft boundaries on the finer-grained jellyfish control task**.

N Limitation and Future Work

There are still several limitations of DiffPhyCon that inspire future work. Firstly, our approach is data-driven, so it is not guaranteed to achieve optimal solutions and also lacks estimation of the error gap between the generated control sequence and the optimal ones. Secondly, the training of DiffPhyCon is currently conducted in an offline fashion, without interaction with a ground-truth solver. Incorporating solvers into the training framework could facilitate real-time feedback, enabling the model to adapt dynamically to the environment and discover novel strategies and solutions. Furthermore, our proposed DiffPhyCon presently operates in an open-loop manner, as it does not consider real-time feedback from solvers. Integrating such feedback would empower the algorithm to adjust its control decisions based on the evolving state of the environment.

O Social Impact Statements

The method we propose offers a means to actively interact with the physical world and achieve specific control objectives. This approach presents significant opportunities for various domains, including fluid control, robotic control, controllable nuclear fusion, and more. However, it is imperative to remain vigilant about potential negative consequences that may arise from this technology to prevent its application to unethical or illegal control issues.

https://doi.org/10.52202/079017-0134

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our paper's contributions and scope are presented in the abstract and introduction accurately.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations of our work are discussed in Section N.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The theoretical result is presented in Section 3.2.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Details of experiments are presented in Appendices D, E, F, G, H, I, J, K, L and M. These details ensure the reproducibility of our results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

https://doi.org/10.52202/079017-0134

Answer: [Yes]

Justification: Our code is available at `https://github.com/AI4Science-WestlakeU/diffphycon`, which also includes the link to our 2D jellyfish movement data.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Details of training and test details are presented in Appendices D, E, F, G, H, I, J, K, L and M. These details ensure the reproducibility of our results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not report error bars because multiple running of our method and all the baselines on the three tasks under different settings are computationally expensive.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Details about the compute resources are provided in Appendix K.4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Societal impacts of our research are discussed in Appendix O.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

https://doi.org/10.52202/079017-0134

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our released data are generated by an open-source simulator. Our paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Our released data are generated by a simulator, which is properly credited and the license and terms of use are explicitly mentioned and properly respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide details about the dataset in the Section of Experiment and Appendices D, F, and G.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.

https://doi.org/10.52202/079017-0134

- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.