From Visual Computing to Physical AI: Simulation, Optimization, and Learning for Physical World

by

Yifei Li

B.S., Carnegie Mellon University (2020)
S.M., Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2026

© 2026 Yifei Li. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Yifei Li
Department of Electrical Engineering and Computer Science
May 12, 2026

Certified by: Wojciech Matusik
Professor of Electrical Engineering and Computer Science, Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

THESIS COMMITTEE

Thesis Supervisor

Wojciech Matusik
Professor of Electrical Engineering and Computer Science
Department of Electrical Engineering and Computer Science

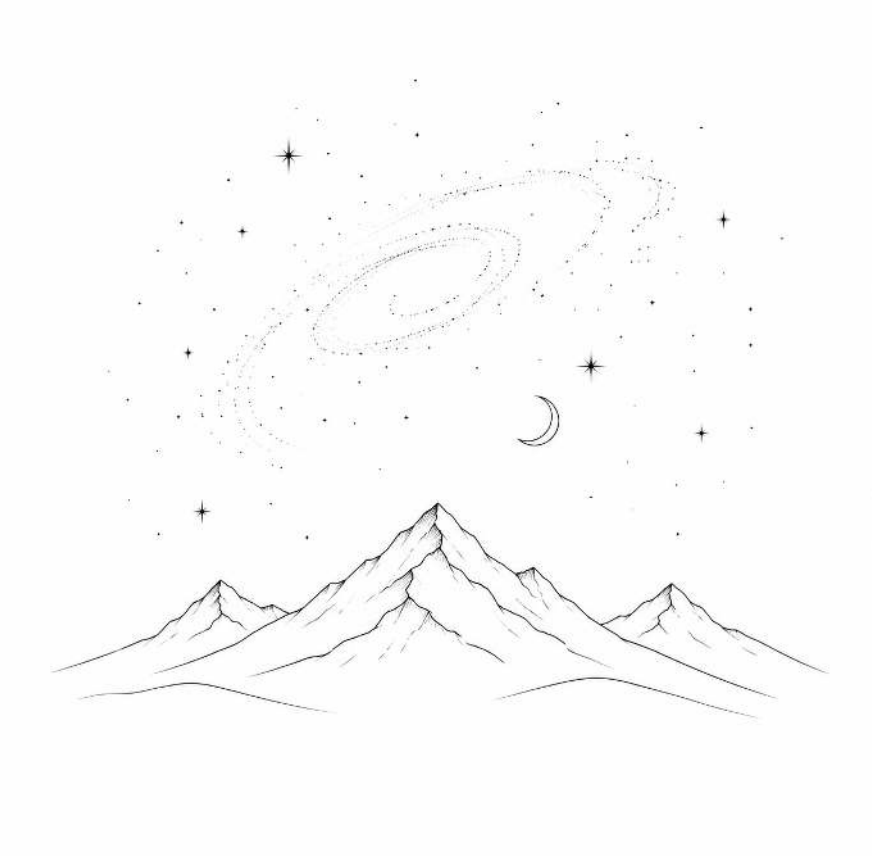
Thesis Readers

Wojciech Matusik
Professor of Electrical Engineering and Computer Science
Department of Electrical Engineering and Computer Science

William T. Freeman
Professor of Electrical Engineering and Computer Science
Department of Electrical Engineering and Computer Science

Jonathan Ragan-Kelley
Associate Professor of Electrical Engineering and Computer Science
Department of Electrical Engineering and Computer Science

To curiosity, and to the courage and persistence to follow it into the unknown.



From Visual Computing to Physical AI: Simulation, Optimization, and Learning for Physical World

by

Yifei Li

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2026 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

This thesis develops computational foundations for Physical AI by transforming physics simulation from a forward prediction tool into a framework for optimization, control, and learning in intelligent physical systems. In traditional computer graphics, the success of simulation is often judged by visual plausibility: if the result appears correct, the task is complete. In domains such as robotics, physical design, and engineering, however, simulation errors have direct physical consequences and can lead to real-world failure. Although high-fidelity simulators exist for such domains, they typically function as forward-only oracles that do not directly provide the gradients needed for inverse design, system identification, or policy optimization. Conversely, purely data-driven models offer fast inference and adaptability from measurements, but often lack the physical structure and consistency required for reliable deployment.

To address this gap, this thesis introduces a framework that bridges physics-based simulation and learning. First, it develops differentiability-aware simulation methods that turn fluid dynamics into usable gradients, enabling optimization and co-design in fluid-solid systems. These methods are instantiated in differentiable topology optimization of fluidic devices and in dynamic fluidic systems where geometry and control are optimized jointly, with applications including flow shaping, fluidic devices, and artificial-heart-inspired systems. Second, the thesis develops structured learning methods that augment, rather than replace, physical simulators, allowing them to adapt to complex real-world dynamics from data while preserving their physical organization. Together, these contributions advance simulation as a computational substrate for reasoning, design, and adaptation, and establish a foundation for intelligent systems that can better understand, predict, and act in the physical world.

Thesis supervisor: Wojciech Matusik
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank the people who guided me most closely through this PhD.

First and foremost, I am deeply grateful to my advisor, Prof. Wojciech Matusik, for his guidance and support over the years. More importantly, he shaped how I think about research: how to choose problems, how to connect ideas across areas, and how to pursue work that is both foundational and impactful. I have learned enormously from his vision, standards, and approach to science.

I am also deeply grateful to Prof. Jessica Hodgins, my undergraduate advisor, who first introduced me to research nine years ago and ultimately set me on the path to a PhD. Without Jessica, I would not have completed this thesis. Her early guidance, encouragement, and belief in me changed the course of my life.

I would also like to thank Prof. Tao Du, Prof. Bo Zhu, Prof. Eftychios Sifakis, and Tuur Stuyck for their mentorship and close advising across different stages of this work. Each of you shaped my PhD in a different way through technical guidance, mentorship, careful feedback on experiments, and help sharpening both the framing of the problems and the evaluation of the results. I am sincerely grateful for your generosity with your time and for continually pushing the work to be more rigorous and more meaningful.

I would like to thank my committee members, Prof. Bill Freeman and Prof. Jonathan Ragan-Kelley, for their time, thoughtful feedback, and support. Bill has also been my academic advisor, and I am grateful for his guidance throughout my PhD. Jonathan first served on my RQE committee and now also serves on my PhD committee. I very much appreciate both of you for being part of this process.

I am also grateful to my labmates, past and present, for the community, friendship, and support they provided over the years. Research is never done alone, and I have been very fortunate to be surrounded by thoughtful, generous, and engaging people throughout this journey.

A special thank you goes to our administrators, Debbie, Megan, Felicia, and Roger, who handled so much of the behind-the-scenes logistics and made it much easier for me to focus on research. Your help has been essential.

I would also like to thank all of my collaborators. I am deeply grateful for the ideas, effort, and time that so many of you contributed. I learned a great deal from working with you. Research is a collective effort, and many of you shaped the final papers in concrete ways through writing and editing, figure design, experiments, and detailed discussions that improved both the results and their presentation.

I also want to thank my friends. They made my PhD years not only manageable, but joyful. I am grateful for the many moments of support, companionship, and happiness that

brightened this journey.

I am especially grateful to Alice for supporting me over the past three years. She helped me through nearly every aspect of my life, including work, family, relationships, surgery, and many fears, anxieties, and uncertainties along the way. Her care and steadiness made an enormous difference in both my PhD and my life.

I want to thank my family for their unconditional love and support throughout my PhD, and for always believing in me, even from far away. I could not have done this without them.

I also want to thank my cats, Yihui and Balanchine. Both came into my life during my PhD and have been a constant source of comfort, companionship, and joy.

I am deeply thankful to my boyfriend, Sifei Luan. I met him in the second half of my PhD, and he supported me through some of the most challenging moments of these years. I am grateful for the happiness he brought into my life, for keeping me grounded, and for always believing in me when I was stressed or uncertain.

Finally, I am grateful for the persistence and growth this journey required of me. Life is full of challenges and uncertainties, and this PhD gave me the resilience and strength to face them. This journey brought many difficulties, uncertainties, and moments when the path forward was not obvious. I am proud that I kept going, kept learning and growing, and kept finding a way through.

Contents

List of Figures 15
List of Tables 21

1 Introduction **23**
1.1 From Visual Computing to Physical Consequence . . . 23
1.2 The Limits of Forward-Only Simulation . . . 24
1.3 The Need for Data Adaptability . . . 25
1.4 Bridging Simulation and Learning . . . 26
1.5 Thesis Scope and Contributions . . . 27
1.6 Thesis Structure . . . 29

2 Related Work **31**
2.1 High-Fidelity Physical Simulation . . . 31
2.2 Differentiable Physics Simulation . . . 31
2.3 Differentiable Design of Flow Systems . . . 32
2.4 Neural Simulators and Hybrid Neural-Physics Methods . . . 33
2.5 Thesis Perspective . . . 34

3 Differentiability-Aware Fluid-Solid Simulation & Optimization **35**
3.1 Motivation . . . 35
3.2 System Overview . . . 37
3.3 Governing Equations . . . 38
3.3.1 Isotropic Stokes Equations . . . 38
3.3.2 Anisotropic Stokes Equations . . . 39
3.4 Numerical Simulation . . . 43
3.4.1 Anisotropic Material Parametrization . . . 43
3.4.2 Discretization Scheme . . . 44
3.4.3 Block divergence . . . 46
3.5 Optimization . . . 47
3.5.1 Problem Statement . . . 48
3.5.2 Numerical Optimizer . . . 49
3.5.3 Choice of Parameters and Interpolation Functions . . . 49
3.6 Applications . . . 50
3.7 Evaluations . . . 53
3.8 Discussion and Conclusion . . . 57

4 Scalable Optimization and Co-Design in Dynamic Fluidic Systems 61
4.1 Overview . . . 61
4.2 Method . . . 62
4.2.1 Pipeline Overview . . . 62
4.2.2 Geometry Representation . . . 63
4.2.3 Differentiable Navier-Stokes Simulation . . . 64
4.3 Benchmarks and Applications . . . 68
4.3.1 Application Overview . . . 69
4.3.2 Scaling to Complex Fluid Fields: Artificial Heart . . . 71
4.4 Experiments . . . 72
4.4.1 Effects of Initialization on Optimization . . . 72
4.4.2 Gradient-Based vs Gradient-Free Optimization . . . 72
4.4.3 Time Performance Profiling and Comparison with PhiFlow . . . 72
4.4.4 Memory and Time Performance Profiling and Comparison with DiffTaichi 73
4.4.5 Fluid Solver Validation – Kármán Vortex Street . . . 74
4.4.6 Gradient Stability and Solver Steps Statistics . . . 74
4.4.7 Gradient Validation . . . 76
4.5 Discussion and Conclusion . . . 76

5 Neural Modular Physics and Data Adaptation 79
5.1 Motivation . . . 79
5.2 Neural Modular Physics . . . 81
5.2.1 Inspirations from Classical Simulators . . . 81
5.2.2 Elastic Physics Modularization . . . 82
5.2.3 Modular Physics Training . . . 84
5.3 Experiments . . . 85
5.3.1 Main Results . . . 86
5.3.2 Comparison with Physics Simulators . . . 88
5.3.3 Unique Benefits of Modularization . . . 89
5.3.4 Ablation Study . . . 90
5.3.5 Failure Mode Analysis . . . 91
5.3.6 Comparison with Baselines . . . 92
5.3.7 Initial Condition Distribution Illustration . . . 93
5.3.8 Generalization to Unseen Initial Conditions . . . 94
5.3.9 Standard Deviations for Table 5.1 . . . 94
5.3.10 Physics Validation Metrics . . . 94
5.4 Implementation Details . . . 95
5.4.1 Groundtruth Simulator . . . 95
5.4.2 Dataset Generation . . . 95
5.4.3 Network architecture and training details . . . 96
5.4.4 Baseline Implementation . . . 97
5.4.5 Interchange Inference Experiment Settings . . . 98
5.5 Discussion and Conclusion . . . 98

6 Conclusion and Future Work **99**
6.1 Future Work . . . 99
6.1.1 Physics Foundation Modules . . . 99
6.1.2 Learning in the Real World . . . 99
6.1.3 Toward Intelligent Physical Systems . . . 100
6.2 Conclusion . . . 101

List of Figures

1.1 From visual plausibility to physical consequence. In traditional visual computing, simulation is often judged by whether the result appears realistic, whereas in domains such as robotics, healthcare, engineering, and digital design, simulation must support physically meaningful reasoning and reliable real-world performance. 24

1.2 Overview of the thesis contributions and their organization around accurate simulation, scalable differentiability, and data adaptability. The three technical components of the dissertation are instantiated through TopoStokes [1], NeuralFluid [2], and Neural Modular Physics [3]. 28

3.1 We present a topology optimization pipeline for designing Stokes-flow fluidic systems with flexible and accurate boundary conditions. Our method automatically creates the structure of this fluidic twister on an $100 \times 100 \times 100$ grid after optimizing nearly four million decision variables. The goal of this device is to generate a swirl flow at its outlet given a constant inflow. Left: our final design is made of spatially-varying anisotropic materials, which we visualize as a small disk in each voxel colored based on its anisotropic direction (bottom-left inset). Our method automatically synthesizes a propeller-like structure (top-right inset) to facilitate the vortex generation near the outlet. Middle: the flow simulated from the final design, visualized as streamlines. A vortex emerges near the outlet (top-right inset). Right: visualization of flows from the fluidic device after 1, 10, 20, and 50 iterations of topology optimization. 36

3.2 An overview of our pipeline: (1) Our system takes as input the specification of a fluidic device design task on a regular grid, including the locations and desired profiles of the Stokes flow at the inlet and outlet; (2) The system then represents the fluidic device using anisotropic materials in each voxel, which we further parametrize using its anisotropic direction (the principal axes of the ellipse in the illustration), viscosity (radius of the ellipse), and impedance for fluid (shown as the background color of the voxel); (3) A numerical differentiable simulator receives the design and solves the Stokes flow; (4) The pipeline then compares the simulated flow at the outlet with the target profile given in the specification and computes a loss function characterizing their discrepancy. The loss is then backpropagated through the numerical simulator to compute its gradients with respect to the anisotropic material parameters. The pipeline runs the method of moving asymptotes (MMA, [4]), a gradient-based optimizer, to improve the design; (5) The pipeline outputs a final design after post-processing the results from a converged optimization process. 37
3.3 An illustration of the anisotropic material model in the quasi-incompressible Stokes flow. We selected three representative material points from the solid phase (orange), the fluid phase (green), and the free-slip solid-fluid boundary (yellow). We associate each material point with an ellipse visualization of $\boldsymbol{K}_f$ in a way that the material inhibits flow along its minor principal axis (small radii). For example, a solid material point forces flow along all directions to be zero and is therefore associated with a small circle. Similarly, a material point on the free-slip boundary impedes flow only in the normal direction, so its $\boldsymbol{K}_f$ is a highly eccentric ellipse aligned with the tangent and normal direction of the boundary. 40
3.4 Designing an *Amplifier* (left) and a *Mixer* (right) under a grid resolution of $80 \times 80 \times 80$. The anisotropic boundaries of the optimized designs are visualized using small colored disks. 50
3.5 Our pipeline generates a tree diffuser on a $80 \times 80 \times 80$ grid. Top left: The anisotropic boundary of the optimized design is visualized using small colored disks. Bottom: The 3 images in the middle visualize the perpendicular cross-sections of the optimized design at different depths from the inlet. These cross-sections highlight the progressive branching from a single inlet to multiple outlets. Top right: The fluid flow, simulated from the optimized design, is visualized as streamlines. 51
3.6 We design the *Circuit* with maximum volume fraction $V_{\max} = 0.25$ (top) and $V_{\max} = 0.5$ (bottom). Inset: We visualize the domain setup. Different faces of the domain are marked as inlet and outlet with different flow velocity. Left: The anisotropic boundary of the generated design is visualized using small colored discs. Right: The resulting fluid flow from the optimized design is visualized using colorcoded streamlines. 53

3.7 Evaluation of our solver on a 2D fluid amplifier design. We visualize the norm of the nodal velocity field simulated in square domains of dimension 32, 256 and 2048 (top left, top middle, bottom left). The relative error (to the solution at dimension 2048) monotonically decreases as the resolution increases. We additionally simulate the same design at 2048x2048 using a traditional Stokes flow solver from Du et al. [5] (bottom middle), and visualize the difference field (bottom right, normalized to inflow value). 54
3.8 Comparisons between our method and [6] using a 2D design problem (60×60 cells). The domain has one inlet and outlet at its left and right, respectively. The goal is to synthesize a structure that transports the inlet flow with velocity $(v_i, 0)$ to the outlet. From left to right: we run both methods to solve this design problem with varying volume fraction limits from 0.6 to 0.2. For both methods, we visualize the optimized fluidity field (row 1, 2) and velocity field (row 4, 5). We additionally visualize the optimized isotropicity field ϵ for our method (row 3). As we decrease the volume fraction from left to right, the method of Borrvall and Petersson [6] starts to synthesize non-physical designs and flow, i.e., the solid cells near the inlet and outlet as well as nonzero fluid velocity on them. 55
3.9 Sensitivity of optimization results to block size. We optimize the 2D amplifier design problem with block sizes 4, 8, and 16 where initial $L_f = 141.938$. We visualize the norm of the velocity field (top) and the optimized isotropy field (bottom) and report the optimized L_f value (bottom). 56
3.10 Solving the Fluidic Twister example using the open-source implementation from Du et al. [5]. Left: The parametric shape and fluid flow visualization before optimization, with the top-right insets showing the designs viewed from their outlets. Right: The corresponding visualizations after optimization. . . 57
3.11 We study the effects of block divergence for simulating divergence-free Stokes flow using this two-inlet, two-outlet design problem in 2D (30×30). The inlets and outlets are defined on the left and right sides of the domain, respectively. Left: We model the design with moderate k_f and λ_0 without imposing the block divergence constraints in simulation. The outflux on the two outlets is 67% of the influx from the two inlets, indicating the flow velocities dissipate into the solid phase inside the domain. Right: We rerun the same simulation with block divergence constraints with block sizes of 8×8 (dashed red lines). The resulting outflux is 100% of the influx. 58

4.1 Pipeline Overview. (1) Our pipeline starts with an initial parametric geometry and a neural network parameterized controller. (2) The fluid dynamics is then simulated using a dynamic Navier-Stokes solver. (3) The performance of the design and control is evaluated using a loss function, the gradients of which are then back-propagated through our end-to-end differentiable framework. (4) The gradient-based optimization iteratively improves the geometry and control to achieve the task goal. This pipeline allows for efficient geometry and control co-optimization. 63

4.2 Applications Overview. In each application, the blue dashed line represents the inlet, the red dashed line indicates the outlet, the white arrows show the flow direction, and the orange shapes and arrows denote the geometry and its motion direction. . . . 69
4.3 (a) Visualization of *Amplifier*. (b) Visualization of *Flow Modulator*. . . . 70
4.4 Fluidic Switch. The switch rotates dynamically across time (dotted lines). The shape of the switch is parameterized as a 2D Polar Bezier, whose parameters, along with the parameters of the rotation signal are subject to optimization. The top and bottom of the illustration visualize information from the initial and optimized iteration respectively. For each iteration, we visualize the design geometry (left) and corresponding streamlines of the flow field (right) at 7 key-frames evenly sampled across time. We additionally plot the target (green) and outlet velocity norm profile (orange) across time and visualize their difference in grey shaded area. . . . 70
4.5 Artificial Heart. *Left*: visualization of the domain and the location of the muscles. *Middle*: Optimized control policy rollout visualization. *Right:* Optimization results visualization. The top and bottom diagrams visualize the cosine and the ECG target variants. . . . 71
4.6 Ablation Studies. *Left:* Optimization trajectories for *Neural Heart* with 7100 parameters under different initialization. Iterations are visualized on a log scale. *Right:* Log scaled loss-iteration curves of our gradient-based method and other gradient-free optimization methods. . . . 74
4.7 Solver Validation. Visualization of Karman Vortex Street under different viscosity conditions. Here, we illustrate the results of the classic Karman Vortex Street test for three different kinematic viscosity values (From top to down $\nu = 0.0$, $\nu = 0.002$ and $\nu = 0.02$), simulated using our differentiable simulator within a domain size 512×1024. Each figure visualizes the vortex patterns, and the results demonstrate how increased viscosity leads to a notable change in vortex formation and dissipation. . . . 75

5.1 NMP modularizes elastic physics simulation into two parts: neural constitutive module to compute internal forces $\mathbf{f}_{\text{int}}$, and neural integration module to evolve system state $\mathbf{s}_t = \{\mathbf{x}_t, \mathbf{v}_t\}$ and handle traditional boundary conditions $\mathbf{S}_{BC}$ and collision $\mathcal{C}$. . . . 81
5.2 Modular training strategy and inference scheme. A two-stage training paradigm is used to optimize neural modules. Separate training in the first stage prevents module collapse, while joint fine-tuning in the second stage further leverages physical constraints. . . . 83
5.3 Long-Horizon Stability and Generalization of Neural Modular Physics. Top: Long-horizon rollouts under random initial conditions, showing the rest mesh and simulation keyframes on all datasets. Ours remains stable and closely matches the ground truth (FEM) over 1,000 steps. Bottom: Generalization to unseen initial conditions and higher resolutions, including unseen initial velocity (Bob), unseen velocity and pose (Cube), and resolution generalization (CubeXL, 10k→21k vertices). . . . 85

5.4 Simulation comparison on FRICTION environment with unknown frictional dynamics. We mark the initial position of duck-surface contact with red circles. 87
5.5 Sim-to-real proof of concept. We pretrain an NMP model on CUBE and finetune it on a scenario with unknown external force and a softer material. To mimic the real-world simulation task, only the position information is accessible during finetuning. 87
5.6 (**a-b**) Comparison between joint training two modular networks and our two-stage independent training strategy in BOB. **(c)** *Impact of physical constraints* on three trajectories in CUBE. The curve records the number of tetrahedra exceeding 5% volume change at each time step and the legend counts the total number violating during 1,000 steps. **(d)** *Interchange inference.* Top: Sequence generated by our trained base model. Middle: Replacing the neural integration module with a semi-implicit integrator with a much smaller timestep. Bottom: Replacing the neural constitutive module with a softer St. Venant material increases deformation, visible in the foot position. 88
5.7 Failure mode analysis. We visualize the last frame in the worst case, which is selected according to the RMSE of NMP. The predictions of other baselines are also included. 91
5.8 Qualitative comparison of rollouts for the finite element ground truth and baselines at 100, 500, and 1000 steps on BOB. From top to bottom, rows show Ground Truth, MeshGraphNet, EGNO, and Transolver. Columns show snapshots at 100, 500, and 1000 steps. 92
5.9 Visualization of trajectories with different initial conditions on CUBE. 93
5.10 Generalization to unseen initial conditions for SPOT. Each row shows a sequence with different initial velocities, demonstrating our model's ability to handle varied dynamics while maintaining fixed-point constraints. Our predictions (blue) closely match the ground truth (green). 94

6.1 Conceptual illustration of an intelligent physical system in aerospace, where data are used to update the internal model and drive both adaptive form and adaptive control under changing flow conditions. 100
6.2 Representative directions for intelligent physical systems, including intelligent metamaterial sensing, soft assistive exoskeletons, and adaptive aeroelastic wings. In each case, sensing, physical structure, and control are integrated within a shared adaptive framework. 101

List of Tables

3.1 We summarize the anisotropic material parameters for modeling different phases and boundary conditions in quasi-incompressible Stokes problem: $\lambda_0 \in \mathcal{R}^+$ represents a predefined scalar parameter controlling the incompressibility in the quasi-incompressible Stokes flow; $k_f \in \mathcal{R}^+$ is a scalar parameter that determines the material's impedance for fluid flow, with $k_f \to +\infty$ creating the solid phase; $\boldsymbol{n} \in \mathcal{R}^d$ is the unit normal vector on the solid-fluid boundary that decides the material's anisotropic responses to normal and tangent flows. 42

3.2 We report the statistics from optimizing the design problems in Sec. 3.6, including the maximum fluidic volume fraction, the functional loss before and after optimization, and the time decomposition in each optimization iteration: "Forward" represents the time spent on computing the loss function, which is dominated by the numerical simulator; "Backprop." stands for the time spent on computing the gradients; "MMA optimizer" reports the time cost from running one iteration of MMA after obtaining the loss and gradients; "Total" is the sum of all the time above. 52

4.1 Application Specifications. We summarize the simulation and optimization configuration for the design applications shown in Sec. 4.3 and report the initial and optimized loss. We note that because our implementation adopts CFL condition for numerical stability during simulation, the actual steps simulated and back-propagated are higher than the numbers shown in "# Frames". . . 69

4.2 Time Performance. Our method achieves one order of magnitude speedup across all resolutions compared to PhiFlow in both forward simulation and backward gradient propagation. 73

4.3 Memory and time performance comparison with DiffTaichi 73

4.4 Statistics of the gradient norm and step count over the full course of optimization for three tasks of varying complexity. 76

4.5 Gradient Validation for Shape Identifier Task 76

5.1 RMSE between model-simulated vertex positions and ground-truth simulations across rollout horizons. See Section 5.3.6 for baseline visualizations. 86

5.2 Inference efficiency comparison on Bob for a 1,000-step rollout. 88

5.3 Physical constraint loss weight analysis. The following test is conducted on the "real-world" Cube scenario in Figure 5.5. 91

5.4 Compare with enhanced baselines, where each model will directly predict 100 steps at one inference. The RMSE of 1,000 steps is recorded. 93
5.5 Standard deviations of RMSE for Table 5.1 (computed over 8 validation trajectories). 95
5.6 Physics validation of different models, where we record the number of tetrahedra exceeding 5% volume change. A smaller number indicates better physical soundness. 95

Chapter 1

Introduction

1.1 From Visual Computing to Physical Consequence

Computer graphics has studied physical simulation for decades, and much of this work has been driven by the goal of visual plausibility. In applications such as film, animation, and games, simulation is often considered successful when the resulting motion or deformation appears convincing to a human observer. Under this standard, the underlying physical model need not be exact, so long as the visual effect is sufficiently realistic. This emphasis has made simulation a central tool in visual computing, enabling the creation of rich and believable dynamic phenomena across a wide range of digital content.

As simulation increasingly moves beyond entertainment, however, the criterion for success changes. In robotics, a policy learned in simulation must transfer to the behavior of real objects and materials under contact [7]. In healthcare, simulation may guide the design of devices operating under strict physiological and flow constraints. In engineering, small inaccuracies in aerodynamic or structural simulation can lead to large deviations in performance [8–10]. In digital design, garments and other physical artifacts created in simulation must behave consistently when realized in the real world [11, 12]. In these settings, it is no longer sufficient for a simulator to produce results that merely look correct. Instead, simulation must support physically meaningful reasoning about systems whose behavior has direct real-world consequences. Figure 1.1 illustrates this shift from visual plausibility to physical consequence across several representative application domains.

This shift in application reveals a fundamental limitation in the dominant role simulation has traditionally played. High-fidelity simulators already exist for many of these domains, including finite element methods for solids [10], computational fluid dynamics for fluids [6, 13], and multibody dynamics for robotic systems [14]. These tools can be highly accurate within their intended regimes, and they remain indispensable for modeling complex physical behavior. Yet they are typically used as forward analysis engines: given a particular geometry, material configuration, or control input, they predict the resulting system behavior. What they do not directly provide is guidance for how to improve that behavior. For design, control, and inference, the central question is not only what happens under a given configuration, but how that configuration should be changed to better satisfy a desired objective. This thesis is motivated by the view that this transition, from visual plausibility to physical consequence,

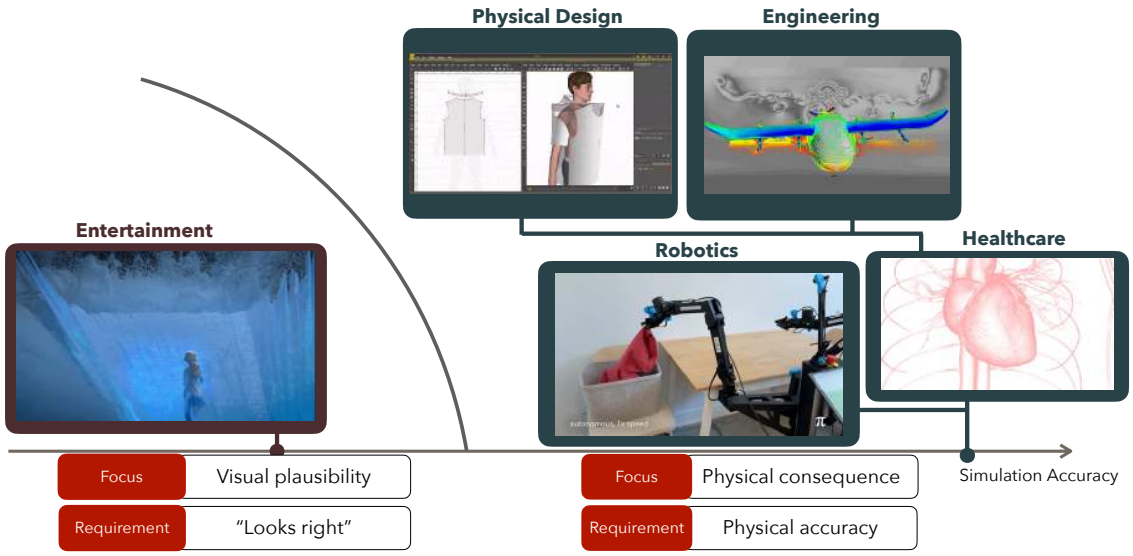


Figure 1.1: From visual plausibility to physical consequence. In traditional visual computing, simulation is often judged by whether the result appears realistic, whereas in domains such as robotics, healthcare, engineering, and digital design, simulation must support physically meaningful reasoning and reliable real-world performance.

requires a broader computational role for simulation. Simulation must become more than a predictive tool. It must serve as a foundation for reasoning, optimization, and adaptation in intelligent physical systems. The chapters that follow develop this perspective by studying how high-fidelity simulation can be made usable for inverse design, control, and learning, while remaining sufficiently accurate to support deployment in the physical world.

1.2 The Limits of Forward-Only Simulation

The increasing importance of physically accurate simulation has been matched by the development of highly sophisticated numerical tools. These simulators are often highly accurate within their intended domains and remain indispensable for analysis and validation. In this sense, the challenge addressed in this thesis is not the absence of physical models, but rather the limited computational role that such models currently play. High-fidelity simulators already exist across the relevant application domains, but they are primarily used as engines for forward prediction rather than as tools for reasoning and optimization.

A forward simulator takes as input a fully specified configuration of the system, such as a geometry, a material distribution, or a control policy, and predicts the resulting behavior. This capability is extremely valuable, but it answers only part of the design problem. In many settings, the central question is not simply what will happen under a given configuration, but how the configuration should be changed so that the outcome better satisfies a desired objective. A robot designer may wish to reduce task failure rate, an engineer may wish to reduce drag, and a biomedical designer may wish to improve the flow characteristics of an artificial heart. A forward simulator can evaluate each candidate design separately, thereby

sampling the performance landscape, but it does not directly reveal the direction in which the design should move to improve performance. This is the sense in which high-fidelity simulators behave as forward-only oracles: they provide accurate evaluations of individual configurations, but not the gradients required for efficient reasoning over the design space.

This limitation becomes particularly severe in high-dimensional problems. Modern physical systems are often parameterized not by one or two variables, but by large collections of geometric, material, and control parameters that interact through nonlinear dynamics. In such regimes, trial-and-error search is computationally inefficient and often impractical. What is needed instead is a way to expose how small changes in the configuration affect the objective, so that optimization can proceed through structured updates rather than blind exploration. Without such gradients, design reduces to trial and error over a large search space rather than principled optimization.

The absence of gradients is therefore not merely a numerical inconvenience; it is a conceptual bottleneck that restricts what simulation can be used for. If physical simulation is to support inverse design, system identification, and control, it must do more than predict outcomes. It must become part of a computational framework that reveals how objectives depend on physical parameters and decisions. This observation motivates the first technical direction of the thesis: to reformulate physically accurate simulation so that it becomes differentiable in the ways required for optimization and learning, while preserving the fidelity needed for real-world physical applications.

1.3 The Need for Data Adaptability

Even when high-fidelity simulators are physically accurate in principle, they are still built on modeling assumptions that are specified in advance. Material laws, contact behavior, boundary conditions, and environmental effects are typically chosen by hand and calibrated for a particular regime [10, 13]. This approach is often sufficient for controlled analysis, but it becomes limiting when simulation is used as part of a larger pipeline for design, control, or deployment. In such settings, the relevant question is not only whether the simulator is accurate under ideal assumptions, but whether it remains accurate when confronted with the variability and complexity of the real world.

In practice, real-world dynamics are often messier than the equations used at simulation time. Materials may exhibit behaviors that are difficult to capture with simplified constitutive models, contact and friction may deviate from idealized laws, and environmental conditions may vary in ways that are hard to specify analytically in advance [15, 16]. As a result, a simulator that is physically grounded and numerically accurate can still produce systematic mismatch when transferred to deployment. This is a second fundamental limitation of classical simulation frameworks: they are not naturally data-adaptive. Once their governing assumptions are fixed, they do not easily incorporate new measurements, new calibrations, or changing operating conditions.

This gap is especially important in the application domains motivating this thesis. In robotics, a policy trained in simulation may fail when real contact dynamics differ from the assumed model [7]. In biomedical design, a simulator may capture the nominal behavior of a device while missing patient-specific or environment-specific effects. In engineering systems,

deployment conditions such as turbulence, wear, or manufacturing variation can introduce discrepancies that are not present in the design-time model [8, 9]. In digital design, garments optimized in simulation may fail to transfer faithfully to real materials and real bodies if the simulator does not reflect the relevant physical variability [11, 12]. In all of these cases, the problem is not simply that simulation is imperfect, but that the model must remain open to correction from data.

This observation motivates the second central requirement of the thesis. If simulation is to serve as a foundation for intelligent physical systems, it must be able to adapt when measurements reveal that the assumed physics is incomplete. A useful computational framework must therefore support not only physical accuracy and optimization, but also systematic adaptation to data. The challenge is to do so without discarding the structure, interpretability, and controllability that make physics-based simulation valuable in the first place. This tension leads directly to the next section, which considers whether learning-based models can provide the missing adaptability and what is gained or lost when physical simulation is replaced by learning.

1.4 Bridging Simulation and Learning

A natural response to the limitations of classical simulation is to turn to learning-based models. Recent learning-based approaches to physical modeling, including physics-informed neural networks, graph neural networks, and neural operators, offer two capabilities that are particularly attractive in the context of intelligent physical systems [17–21]. First, because these models are typically differentiable end-to-end, they naturally support gradient-based optimization and learning. Second, because they are trained from data, they can adapt to new measurements, calibrations, and operating conditions in ways that are difficult to achieve with fixed analytical models [15, 16, 22]. These properties make learning-based methods an appealing candidate for addressing the two limitations identified in the preceding sections: the lack of usable gradients and the lack of data adaptability.

However, these advantages come with an important tradeoff. Although learning-based models are flexible and expressive, they often lack the physical consistency, robustness, and controllability required for high-stakes physical applications. A neural model may interpolate well within the regime represented in its training data, yet behave unpredictably outside that regime. More fundamentally, many end-to-end neural simulators behave as monolithic functions whose primary accessible quantity is the model output itself, leaving little or no access to intermediate physical quantities such as internal forces, stress, or deformation measures. This opacity makes it difficult to impose known physical laws, interpret failure modes, or systematically vary physically meaningful parameters. Although physics-informed neural networks attempt to incorporate governing equations directly, they often suffer from optimization difficulties and are typically more successful on relatively simple equations and settings than on the long-horizon, high-complexity dynamics relevant to this thesis [17, 18, 23–26].

The central challenge, therefore, is not to choose between simulation and learning, but to combine their complementary strengths. High-fidelity simulators provide physically grounded models whose parameters, intermediate quantities, and outcomes can be interpreted and

controlled. Learning-based models provide differentiability and adaptation from data. Neither paradigm alone fully supports the needs of intelligent physical systems, but together they suggest a more powerful computational framework: one that is accurate enough to trust, yet also differentiable and adaptive enough to optimize and deploy in the real world. This is the guiding perspective of the present thesis.

The approach developed in this dissertation is to bridge these two worlds through three connected directions. The first is to develop accurate and differentiable physical simulation in challenging regimes where standard formulations do not naturally yield stable or useful gradients, such as frictional contact and fluid-solid coupling. The second is to derive scalable methods for gradient computation, so that physically grounded models can be used efficiently in optimization and learning loops. The third is to introduce structured forms of data adaptation, in which learned components are incorporated into physical simulation without giving up the simulator's underlying organization. Together, these directions aim to establish a broader framework for Physical AI, in which simulation, optimization, and learning are unified to support reasoning, design, and adaptation in the physical world.

1.5 Thesis Scope and Contributions

This thesis develops computational foundations for intelligent physical systems by bridging physics-based simulation, optimization, and learning. The central premise is that simulation should not be treated solely as a tool for forward prediction, nor should learning be treated solely as a replacement for physical modeling. Instead, the thesis advances a unified framework in which high-fidelity simulation provides physical structure and reliability, differentiable computation enables optimization and control, and learning introduces adaptation where analytical models are incomplete. Under this perspective, the goal is to make physical models not only accurate enough to trust, but also usable for reasoning, design, and deployment. The main contributions of this thesis are as follows:

1. **Differentiability-aware physical simulation.** The thesis develops differentiability-aware formulations for physically accurate simulation in regimes where gradients are difficult to obtain or unstable in practice. Two such regimes are central to the dissertation: contact-rich deformable systems and fluid-solid systems. In both cases, the challenge is not merely to simulate the forward dynamics accurately, but to formulate the underlying physical model so that it remains usable within optimization and learning pipelines. This includes, for example, handling frictional contact in deformable simulation and treating interface conditions, topology, and boundary behavior as variables in fluid-solid design. By addressing these regimes, the thesis establishes the physical modeling foundation required for subsequent reasoning tasks.

2. **Scalable optimization through differentiable simulation.** The thesis develops scalable methods for optimization through differentiable simulation. In high-dimensional physical systems, it is not enough for gradients to exist in principle; they must also be computable efficiently enough to support repeated optimization over geometry, materials, latent physical parameters, and control policies. The thesis therefore studies

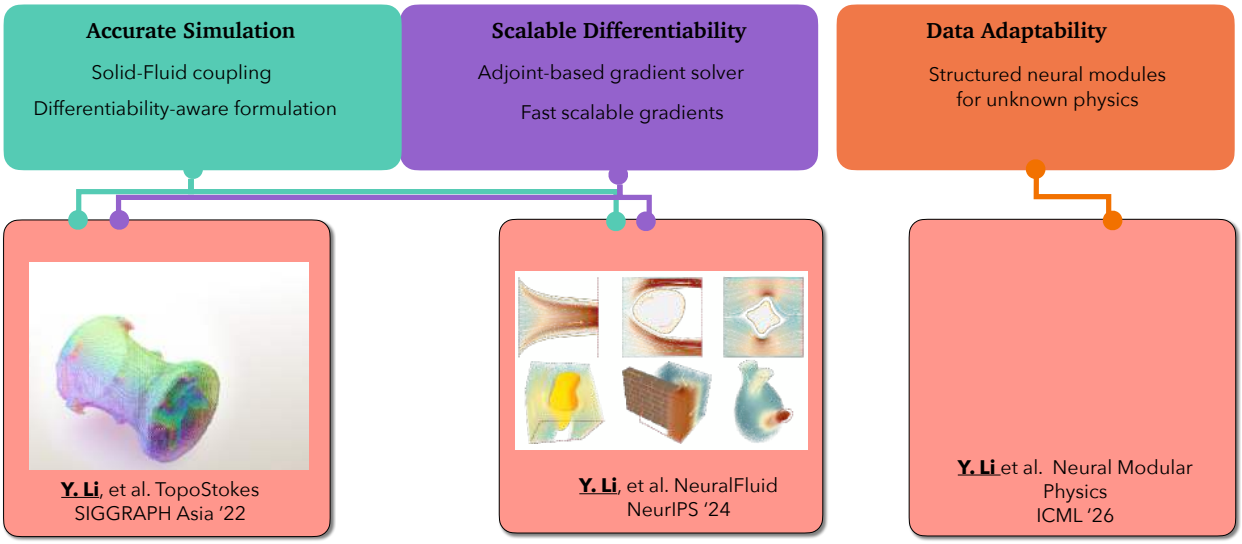


Figure 1.2: Overview of the thesis contributions and their organization around accurate simulation, scalable differentiability, and data adaptability. The three technical components of the dissertation are instantiated through TopoStokes [1], NeuralFluid [2], and Neural Modular Physics [3].

adjoint-based and related computational strategies that make gradients through high-fidelity simulation practical at useful problem scales. This transforms simulation from a forward-only oracle into a computational framework that supports inverse design, system identification, co-design of form and control, and policy learning.

3. **Structured data adaptation in physical simulation.** The thesis develops structured mechanisms for adapting physical models to real-world data. Even accurate and differentiable simulators remain limited by the assumptions built into their governing equations. To address this, the thesis introduces neural modular approaches in which learned components are integrated into well-defined parts of a physical solver while preserving the overall organization of simulation. This enables the model to adapt where constitutive behavior, contact response, or boundary effects are uncertain, without discarding the structure, and controllability that make simulation valuable.

These contributions are summarized in Fig. 1.2. They are developed and demonstrated in the context of fluid-solid design, dynamic fluidic systems, and structured data adaptation in physical simulation. Collectively, they support the broader claim of the dissertation: that progress toward Physical AI requires a synthesis of physical accuracy, differentiability, and data adaptability within a common computational framework. The remainder of the thesis develops this claim through concrete technical contributions in differentiability-aware simulation for fluid-solid systems, scalable optimization and co-design in dynamic flow, and neural modular physics for structured data adaptation.

1.6 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 presents the background and positions the dissertation within prior work on high-fidelity physical simulation, differentiable simulation, and learning-based physical modeling. It also introduces the central perspective of the thesis: that intelligent physical systems require a synthesis of physical accuracy, differentiability, and data adaptability. Chapter 3 develops differentiability-aware simulation for fluid-solid systems in the static setting, with emphasis on interface-aware modeling, topology optimization, and the treatment of material and boundary behavior within a unified differentiable design space. Chapter 4 extends this framework to dynamic fluid-solid systems, focusing on joint reasoning over geometry and control in time-varying flow problems. Chapter 5 introduces Neural Modular Physics as a structured approach to data adaptation, showing how learned components can augment physical solvers while preserving their organization and computational interfaces. Finally, Chapter 6 concludes the thesis by summarizing the main contributions, discussing the broader implications for Physical AI, and outlining promising directions for future work.

Taken together, these chapters develop the central argument of the dissertation: that progress toward intelligent physical systems requires simulation frameworks that are not only physically accurate, but also usable for optimization and adaptable to real-world data. Across fluid-solid design, dynamic flow control, and modular data-adaptive simulation, the thesis advances a unified computational perspective in which simulation, optimization, and learning serve as complementary components of physical reasoning and design.

Chapter 2

Related Work

2.1 High-Fidelity Physical Simulation

High-fidelity physical simulation has a long history across solid mechanics, fluid dynamics, contact, and coupled multiphysics systems. In solid and deformable simulation, continuum-based methods have enabled detailed modeling of elastic bodies, cloth, shells, and contact-rich systems [27–29]. In fluid systems, methods for Stokes and Navier–Stokes flow, together with topology optimization and fluid-structure interaction techniques, have made it possible to analyze and design increasingly complex flow phenomena under physical constraints [6, 13, 30–34]. These methods remain indispensable because they encode known physical laws, respect established numerical structure, and provide interpretable intermediate quantities that support analysis and validation. However, their dominant role remains forward analysis: given a geometry, material distribution, or boundary condition, they predict the resulting behavior, but they are less naturally suited to inverse reasoning, control, or adaptation.

This distinction is important for the present thesis. In many design and control problems, the central question is not only whether a simulator can accurately predict the behavior of a fixed system, but whether it can support reasoning over how that system should be changed. Classical high-fidelity methods are powerful precisely because they are physically grounded, but they are often built around formulations and workflows that prioritize forward solution rather than optimization or learning. As a result, although they provide the physical backbone for the problems studied in this dissertation, additional computational machinery is needed to make them usable for gradient-based design, co-design, and adaptation from data.

2.2 Differentiable Physics Simulation

Differentiable physics simulation has emerged as a powerful framework for turning physical models into computational tools for optimization and learning. Rather than using simulation only for forward prediction, these methods expose gradients of physical objectives with respect to geometry, materials, states, or control variables, enabling inverse design, system identification, and policy optimization. Representative examples include differentiable programming environments such as DiffTaichi [35], as well as differentiable formulations for fluids [36–38], position-based dynamics [39], deformable objects [40], articulated systems

[41, 42], and solid-fluid coupling [43, 44]. These works demonstrate the broader promise of simulation as a substrate for optimization and control across graphics and robotics.

At the same time, existing differentiable simulation methods do not fully address the challenges central to this thesis. In many important regimes, differentiability must shape the physical formulation itself rather than be added afterward. Contact, boundary conditions, topology change, and coupled multiphysics behavior can all make gradients unstable, expensive, or poorly aligned with the design variables of interest. This is especially true in fluid-solid systems, where interface behavior and geometry are often inseparable from the governing model. More broadly, progress in differentiable simulation requires not only gradient computation, but differentiability-aware formulations that remain accurate while exposing stable and useful gradients for optimization and learning.

A second limitation is that differentiability alone does not guarantee practical optimization. In large-scale physical systems, gradients may exist formally while remaining too expensive or too noisy to support repeated design updates. This is particularly relevant when the design variables control geometry, boundary behavior, or dynamic control policies, since the associated optimization loops may require many forward and backward solves. For this reason, the present thesis treats differentiability not as an isolated property, but as part of a larger design problem that includes physical formulation, numerical stability, and computational scalability.

2.3 Differentiable Design of Flow Systems

The optimization of flow systems has a long history in computational engineering, beginning with the pioneering work of Borrvall and Petersson [6] and expanding into a broad literature on topology optimization for fluidic problems [13]. In these approaches, one seeks a geometry or spatial distribution of solid and fluid regions that optimizes a flow objective subject to the governing equations. Applications span Stokes flow [6, 45–47], steady and unsteady flow [30, 31], fluid-structure and fluid-thermal interaction [32–34, 48, 49], microfluidics [50], and aerodynamic design [8, 9]. These works demonstrate the power of simulation-driven optimization for discovering functional flow systems whose geometry would be difficult to specify manually.

This line of work is closely related to the broader paradigm of topology optimization, which has enabled the design of high-performance structures across a wide range of engineering applications [51–53]. In computer graphics, topology optimization has likewise been used for computational fabrication and digital design, including elastic structures [54], shells [55], porous materials [56], and microstructures [57–59]. These works highlight the value of optimization frameworks that can automatically discover geometry rather than restricting design to a low-dimensional parameterization.

Despite this progress, flow optimization remains challenging because the key design variables are tightly coupled to the simulation model itself. Compared with structural optimization, flow design requires careful treatment of fluid-solid interfaces, boundary conditions, and coupled physics, all of which affect both physical fidelity and optimization tractability. In computer graphics, Du et al. [5] developed a differentiable framework for Stokes flow design with different boundary conditions and demonstrated several compelling examples,

but the resulting designs remained restricted to a predefined parameterization. More broadly, this literature points to a central need addressed in this thesis: physical simulation and optimization must be developed together, so that topology, interface behavior, and boundary conditions can be treated as differentiable design variables rather than fixed assumptions.

This issue becomes even more pronounced in dynamic fluidic systems. In static topology optimization, the objective is typically defined over a steady-state flow field, whereas in dynamic settings the design must also account for time-varying actuation, moving boundaries, and control-dependent behavior. This shifts the problem from geometry optimization alone to the joint reasoning over geometry, physics, and control. The fluid-related chapters of this thesis build on the design literature above, but focus specifically on formulations that make such joint optimization possible while preserving physical accuracy at the fluid-solid interface.

2.4 Neural Simulators and Hybrid Neural-Physics Methods

Neural approaches to physical simulation have been widely explored as a means of improving differentiability and data adaptability [19, 21, 22]. Existing methods can be broadly divided into two classes: end-to-end neural simulators and hybrid neural-physics methods. End-to-end neural simulators use a monolithic learned model to approximate system dynamics directly from data. Representative examples include convolutional architectures for Eulerian systems [60–63], graph neural networks for complex geometries and interactions [19, 64–67], and neural operators for predicting physical processes across resolutions and boundary conditions [21, 68–70]. These methods are flexible and data-adaptive, but because they model the full system through a single learned function, they often make it difficult to preserve physical structure, impose known constraints, or access intermediate physical quantities.

Hybrid neural-physics methods instead incorporate learning into selected parts of a classical solver, preserving a physical backbone while using data to model effects that analytical formulations miss. Examples include learned discretization stencils [15], learned residual dynamics [16], and learned constitutive laws in elastic simulation [22]. These methods retain more physical structure than end-to-end models, but they typically modify only isolated components of the pipeline. As a result, they often improve local model fidelity without fundamentally changing the overall computational role of simulation.

A related line of work studies modular neural architectures more broadly, where complex computation is decomposed into specialized components rather than represented by a single monolithic network. Modular neural networks have been explored in language reasoning, visual question answering, and related domains [71, 72]. More generally, modularity can improve interpretability, specialization, and compositional reuse, although its effectiveness depends on whether the modules align with meaningful subproblems [73, 74]. In the context of physical simulation, this perspective suggests that learning should not simply replace a solver wholesale, but should respect the internal structure of numerical simulation pipelines.

Taken together, this literature points to a central need addressed in this thesis: data adaptation should augment physical simulation without discarding its organization and interfaces. The goal is not merely to produce a more flexible predictor, but to maintain access

to intermediate quantities, physically meaningful variables, and computational structure. This motivates the modular perspective developed later in the dissertation, in which learned components are introduced at well-defined stages of the simulation pipeline so that physical structure, interpretability, and controllability are preserved while enabling adaptation to data.

2.5 Thesis Perspective

The literature reviewed above suggests that no single paradigm is sufficient for intelligent physical systems. High-fidelity simulation provides physical structure and accuracy, but is often limited to forward analysis. Differentiable simulation enables optimization and control, but remains challenging in regimes where the formulation itself must be redesigned to expose useful gradients. Learning-based models provide flexibility and adaptation from data, but often sacrifice the structure and controllability that make simulation valuable.

This thesis takes the view that Physical AI requires simulation frameworks that combine physical accuracy, differentiable optimization, and data-adaptive learning, so that simulators become not only tools for forward prediction, but substrates for reasoning, design, control, and adaptation in the physical world. The chapters that follow develop this perspective through three connected technical directions. Chapter 3 studies differentiability-aware fluid-solid simulation in the static setting, with emphasis on interface-aware modeling and topology optimization. Chapter 4 extends this perspective to dynamic fluidic systems, where geometry and control must be optimized jointly. Chapter 5 introduces a modular approach to data adaptation, showing how learned components can augment physical solvers while preserving their structure and interfaces. Together, these chapters instantiate the broader thesis claim that simulation, optimization, and learning must be developed as parts of a common computational framework for intelligent physical systems.

Chapter 3

Differentiability-Aware Fluid-Solid Simulation & Optimization

3.1 Motivation

Fluidic systems play a vital role in modern engineering, supporting applications ranging from jet engines and hydraulic actuators to heart valves and bioreactors. The computational design of a fluidic system with precise functionality and complex geometry remains a substantial challenge. Despite the rapid growth of additive manufacturing, which has enabled the fabrication of intricate flow-driven systems at unprecedented resolution, the computational exploration of the design space of even a simple flow "twister" (one that converts a laminar input flow to a swirling pattern at its outlet) remains difficult due to the intertwined challenges of flow simulation, topological structure exploration, and accurate boundary representation.

This chapter focuses on two underlying challenges in differentiability-aware design of fluidic systems. First, an accurate flow simulator must enforce both incompressibility and boundary conditions, including slip and no-slip behavior, within a topologically complex domain. In particular, when a fluid structure becomes narrow, accurately capturing the impact of the solid boundary on near-boundary flow behavior is essential both for evaluating system performance and for computing meaningful design sensitivities. Conventional approaches based on a strict non-slip boundary condition, which is itself a modeling hypothesis rather than a direct physical principle, can effectively clog such narrow pathways and hinder optimization from discovering useful design features.

Second, a capable optimization framework must effectively explore a high-dimensional design space over both *shape* and *topology* without being constrained by strong parameterization priors. Traditional shape optimization frameworks, although able to represent local boundary geometry accurately, cannot easily generate new topological features that differ substantially from the current design. These two issues together make representation a central problem: the design representation must balance geometric expressiveness, needed for accurate boundary modeling, with topological flexibility, needed for open-ended structural exploration.

Existing fluidic design methods can be broadly grouped into *field-based* and *shape-based* approaches, according to how the design is represented. Field-based methods, such as density-

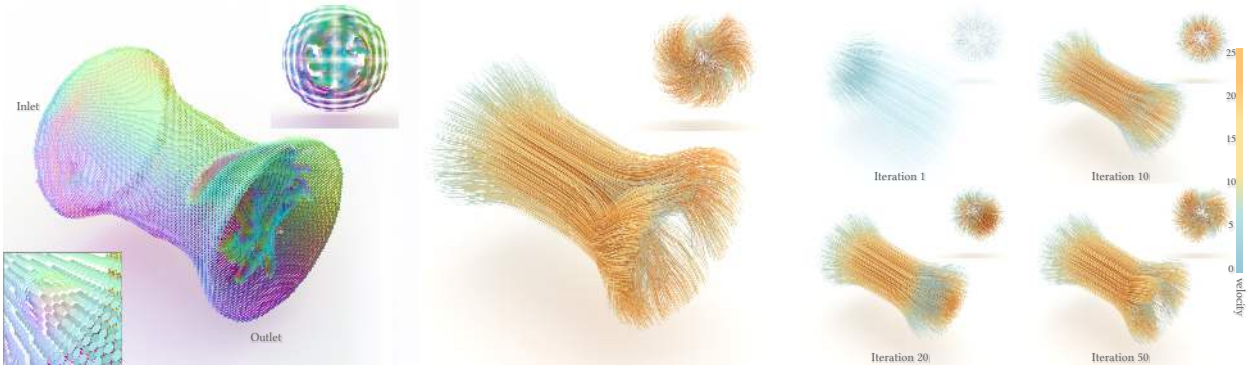


Figure 3.1: We present a topology optimization pipeline for designing Stokes-flow fluidic systems with flexible and accurate boundary conditions. Our method automatically creates the structure of this fluidic twister on an $100 \times 100 \times 100$ grid after optimizing nearly four million decision variables. The goal of this device is to generate a swirl flow at its outlet given a constant inflow. Left: our final design is made of spatially-varying anisotropic materials, which we visualize as a small disk in each voxel colored based on its anisotropic direction (bottom-left inset). Our method automatically synthesizes a propeller-like structure (top-right inset) to facilitate the vortex generation near the outlet. Middle: the flow simulated from the final design, visualized as streamlines. A vortex emerges near the outlet (top-right inset). Right: visualization of flows from the fluidic device after 1, 10, 20, and 50 iterations of topology optimization.

based topology optimization [6], represent the fluid domain on a background grid and provide substantial flexibility for topological exploration. However, like related volume-fraction formulations such as the volume-of-fluid method [75], they face ambiguity in reconstructing sharp interfaces and therefore have difficulty enforcing accurate boundary conditions. Shape-based approaches, including implicit level sets [76] and explicit parametric representations [5], can model boundaries more accurately but are much more limited in their ability to explore large topological changes. In particular, operations such as merging, splitting, or introducing new holes are difficult to represent within a fixed parameterization. A representation that combines the strengths of both field-based and shape-based methods, despite related successes in forward simulation such as CLSVOF [77] and particle level sets [78], remains largely unexplored in the computational design of fluidic systems.

To address these challenges, this chapter develops a non-parametric topology optimization framework with accurate boundary treatment for large-scale fluidic system design. The central idea is to use a unified geometric representation that simultaneously encodes phase (solid or fluid), sharp interface information, and anisotropy. Inspired by anisotropic material models for elastic simulation [79] and tensor-based representations in magnetic resonance imaging [80], the method represents solid regions, fluid regions, and their boundaries as an anisotropic tensor field discretized on a background grid. This representation enables both accurate treatment of boundary conditions and flexible topological evolution during optimization. Building on this formulation, the chapter further develops a differentiable simulation and optimization framework, together with a block-based incompressibility constraint, to support design in a high-dimensional parameter space. Compared with prior flow optimization methods, this framework enables topologically complex flow design by expressing spatially

distributed, multi-phase, and heterogeneous material behavior in a continuous and unified form.

The resulting framework supports computational design of fluidic systems with up to millions of design parameters and enables a range of complex fluidic device designs, including examples with intricate solid structures and free-slip flow behavior in complex fluid domains. More broadly, this chapter provides the first technical component of the thesis by showing how fluid-solid systems can be formulated in a differentiability-aware manner so that simulation remains both physically meaningful and usable for optimization.

3.2 System Overview

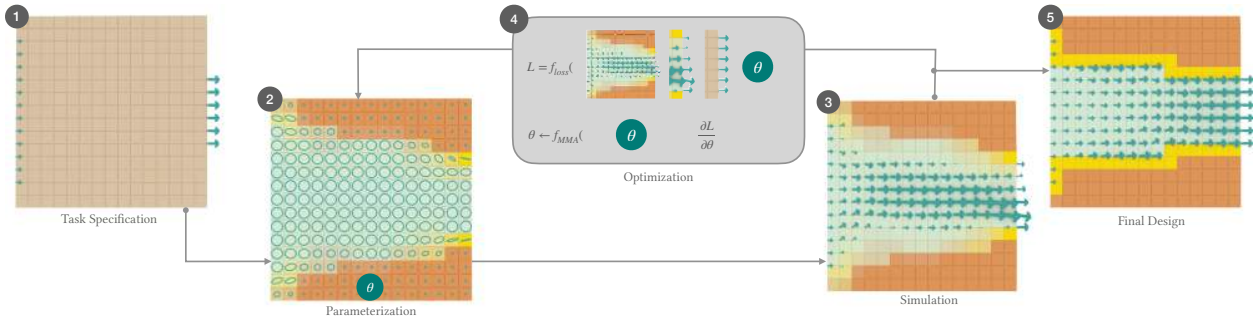


Figure 3.2: An overview of our pipeline: (1) Our system takes as input the specification of a fluidic device design task on a regular grid, including the locations and desired profiles of the Stokes flow at the inlet and outlet; (2) The system then represents the fluidic device using anisotropic materials in each voxel, which we further parametrize using its anisotropic direction (the principal axes of the ellipse in the illustration), viscosity (radius of the ellipse), and impedance for fluid (shown as the background color of the voxel); (3) A numerical differentiable simulator receives the design and solves the Stokes flow; (4) The pipeline then compares the simulated flow at the outlet with the target profile given in the specification and computes a loss function characterizing their discrepancy. The loss is then backpropagated through the numerical simulator to compute its gradients with respect to the anisotropic material parameters. The pipeline runs the method of moving asymptotes (MMA, [4]), a gradient-based optimizer, to improve the design; (5) The pipeline outputs a final design after post-processing the results from a converged optimization process.

We present an overview of the method in Fig. 3.2. The input is a specification of a fluidic device given by target inlet and outlet flow profiles. The device is represented on a regular grid whose voxels are filled with anisotropic materials. These materials are parameterized by scalar fields describing anisotropy, viscosity, and flow impedance. The resulting material distribution defines a multi-phase fluidic device, whose solid-fluid boundaries can be extracted from cells with highly anisotropic materials. A differentiable numerical simulator is then used to compute the Stokes flow of the current design, and the simulated outlet flow is compared with the target profile to evaluate performance. Gradients of the performance metric with respect to the material parameters are obtained by backpropagating through

the simulator. The optimization then uses MMA [4], a standard gradient-based method in topology optimization, to improve the design by evolving its anisotropic material distribution. After convergence, a post-processing step extracts the final design surface.

The remainder of this chapter is organized as follows. Section 3.3 introduces the governing equations of the Stokes flow model. Section 3.4 describes the discretization and numerical solver. Section 3.5 formulates the fluidic design problem as a numerical optimization problem and presents the optimization algorithm. Section 3.6 and Section 3.7 present applications and evaluation, and Section 3.8 concludes the chapter.

3.3 Governing Equations

This section describes the physical model of the Stokes flow problem used in this chapter. While Stokes equations have been extensively studied for decades, we revisit this problem with a focus on developing a novel anisotropic constitutive model that jointly represents different phases (solid and fluid) and boundary conditions (no-slip and free-slip) in a unified manner. Our constitutive model provides a uniform, grid-friendly parametrization of the design space of fluidic devices without sacrificing the flexibility and accuracy in solid-fluid boundary conditions.

3.3.1 Isotropic Stokes Equations

Quasi-incompressible Stokes flow We briefly review the quasi-incompressible Stokes flow model described in Du et al. [5]. Consider a problem domain $\Omega \subset \mathcal{R}^d$ ($d = 2$ or 3). The velocity field $\boldsymbol{v} : \Omega \to \mathcal{R}^d$ of the quasi-incompressible Stokes flow is given by the following energy minimization problem:

$$\min_{\boldsymbol{v}} \quad \int_\Omega \mu \|\nabla \boldsymbol{v}\|_F^2 d\boldsymbol{x} + \int_\Omega \lambda (\nabla \cdot \boldsymbol{v})^2 d\boldsymbol{x}, \tag{3.1}$$

$$s.t. \quad \boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{v}_D(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \partial\Omega_D. \tag{3.2}$$

$$\boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \partial\Omega_F. \tag{3.3}$$

Note that Eqn. (3.1) excludes the external-force energy defined in Du et al. [5] because we assume no external forces (e.g., gravity) in our design problem. Here, the notation $\|\cdot\|_F$ is the Frobenius norm of a matrix, and $\mu \in \mathcal{R}^+$ and $\lambda \in \mathcal{R}^+$ are two scalar parameters denoting the flow's dynamic viscosity and incompressibility, respectively. In particular, $\lambda \to +\infty$ implies perfectly incompressible Stokes flow. The problem considers the following boundary conditions defined on a partition of the domain boundary $\partial\Omega = \partial\Omega_D \cup \partial\Omega_F \cup \partial\Omega_O$: The *Dirichlet boundary* condition specifies a desired velocity profile $\boldsymbol{v}_D$ on the boundary $\partial\Omega_D$, which is either from prescribed inlet flow profiles or from *no-slip boundary* conditions ($\boldsymbol{v}_D = \boldsymbol{0}$); the *free-slip boundary* condition defined on $\partial\Omega_F$ requires the velocity's projection along the normal direction $\boldsymbol{n}$ be zero; finally, the *open boundary* on $\partial\Omega_O$ imposes no explicit constraints on the velocity and automatically satisfies zero-traction conditions once the energy in Eqn. (3.1) is minimized, which is suitable for modeling free flows at an outlet of a fluidic system.

Although we do not consider external forces or non-zero-traction boundary conditions in our problem, they can be accommodated in a similar way described in Du et al. [5].

We refer readers to Du et al. [5] for a comprehensive discussion on the derivation of this quasi-incompressible Stokes flow model and its numerical benefits in fluidic device design problems. While their chapter presented a computational design pipeline for fluidic devices and demonstrated examples with moderately sophisticated solid structures, the method constrained the designs in the space of parametric shapes, which inhibits topologically different designs from emerging.

3.3.2 Anisotropic Stokes Equations

The challenges in previous methods motivate us to develop a new geometric representation that simultaneously accommodates expressive topology, flexible boundary conditions, and accurate simulation in Stokes-flow fluidic systems. Noting that fluid near solid-fluid boundaries satisfies different physical constraints in the normal and tangent directions, we propose an anisotropic material model that uniformly represents solid, fluid, and solid-fluid boundaries, which we describe in detail below.

Anisotropic, quasi-incompressible Stokes flow We propose the following energy minimization problem that modifies the previous isotropic Stokes flow:

$$\min_{\boldsymbol{v}} \quad E_{m,\mu}[\boldsymbol{v}] + E_{m,\lambda}[\boldsymbol{v}] + E_f[\boldsymbol{v}], \tag{3.4}$$

$$s.t. \quad \boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{v}_D(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \partial\mathcal{B}_D. \tag{3.5}$$

where each energy component is defined as follows:

$$E_{m,\mu}[\boldsymbol{v}] := \int_{\mathcal{B}} \mu \|\nabla \boldsymbol{v} \boldsymbol{K}_m^{\frac{1}{2}}(\boldsymbol{x})\|_F^2 d\boldsymbol{x}, \tag{3.6}$$

$$E_{m,\lambda}[\boldsymbol{v}] := \int_{\mathcal{B}} \lambda(\boldsymbol{x})(\nabla \cdot \boldsymbol{v})^2 d\boldsymbol{x}, \tag{3.7}$$

$$E_f := \int_{\mathcal{B}} \|\boldsymbol{K}_f^{\frac{1}{2}}(\boldsymbol{x})\boldsymbol{v}\|_2^2 d\boldsymbol{x}, \tag{3.8}$$

where we use the subscripts m and f in the energy names to indicate they model the material and the frictional effects, respectively. Note that this formulation incorporates the standard, isotropic Stokes model as a special case, namely by setting $\boldsymbol{K}_m = \mathbf{I}$ and $\boldsymbol{K}_f = \mathbf{0}$. The new energy minimization problem introduces a few critical modifications to the original problem in Eqns. (3.1-3.3): First, we change the problem domain from Ω to $\mathcal{B} \subset \mathcal{R}^d$, which we assume to be an axis-aligned, sufficiently large box that encloses the fluidic region Ω. Second, we introduce two symmetric positive semi-definite matrix fields $\boldsymbol{K}_m, \boldsymbol{K}_f : \mathcal{B} \to \mathcal{S}_+^d$ and replace the scalar parameter λ with a spatially varying field $\lambda : \mathcal{B} \to \mathcal{R}^+$. These three new fields define a new material model that enables anisotropic responses to velocities at different directions. Finally, with the domain changing from Ω to a box-shaped $\mathcal{B}$, we adjust the boundary conditions as follows: We consider a partition of the boundary $\partial\mathcal{B}$ into $\partial\mathcal{B} = \partial\mathcal{B}_D \cup \partial\mathcal{B}_O$ where $\partial\mathcal{B}_D$ and $\partial\mathcal{B}_O$ represent the locations of the Dirichlet boundary and the open boundary conditions, respectively. The Dirichlet boundary $\mathcal{B}_D$ now consists of the inlet of the fluidic system where we enforce a prescribed flow profile and the border of the

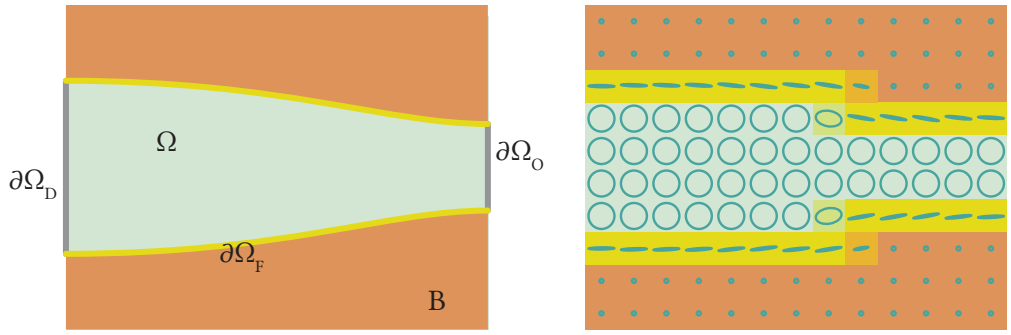


Figure 3.3: An illustration of the anisotropic material model in the quasi-incompressible Stokes flow. We selected three representative material points from the solid phase (orange), the fluid phase (green), and the free-slip solid-fluid boundary (yellow). We associate each material point with an ellipse visualization of $\boldsymbol{K}_f$ in a way that the material inhibits flow along its minor principal axis (small radii). For example, a solid material point forces flow along all directions to be zero and is therefore associated with a small circle. Similarly, a material point on the free-slip boundary impedes flow only in the normal direction, so its $\boldsymbol{K}_f$ is a highly eccentric ellipse aligned with the tangent and normal direction of the boundary.

solid phase, on which we directly assign zero velocities. The open boundary $\partial\mathcal{B}_O$ still models a zero-traction, free-flow outlet like before. The new formulation of boundary conditions does not mean we exclude no-slip or free-slip solid-fluid boundary conditions in our problem, however. In fact, solid-fluid boundaries are now absorbed into the interior of $\mathcal{B}$ and will be represented by a careful choice of $\boldsymbol{K}_m$, $\boldsymbol{K}_f$, and λ. We illustrate the new Stokes flow model in Fig. 3.3.

The new energy minimization problem uses a single material model characterized by spatially-varying $\boldsymbol{K}_m$, $\boldsymbol{K}_f$, and λ in the new problem domain $\mathcal{B}$. We stress that this material model design is not arbitrary but inspired by strong physics intuition. Below, we will discuss the subtleties in these three parameters by demonstrating their capability of expressing different phases (solid and fluid) and various boundary types (no-slip and free-slip). Concretely speaking, we will show that by properly setting them everywhere in $\mathcal{B}$, we can draw an analogy between the two physical models described in Eqns. (3.1-3.3) and Eqns. (3.4-3.5).

Fluid-phase material For any point $\boldsymbol{x}$ in the interior of the fluid phase in Eqns. (3.1-3.3), i.e., $\boldsymbol{x}$ belongs to the interior of Ω, we choose $\boldsymbol{K}_m = \boldsymbol{I}$, $\boldsymbol{K}_f = \boldsymbol{0}$, and $\lambda = \lambda_0$ where λ_0 indicates the scalar parameter used in the original quasi-incompressible Stokes problem (Eqn. (3.1)). This way, the energy in Eqns. (3.6-3.7) becomes identical to Eqn. (3.1), confirming that it preserves the physics model of quasi-incompressible Stokes flow in the fluid phase.

Solid-phase material Similarly, for any point $\boldsymbol{x}$ in the interior of the solid phase, we set $\boldsymbol{K}_f = k_f\boldsymbol{I}$ where $k_f \to +\infty$. According to the energy in Eqn. (3.8), this will force the velocity $\boldsymbol{v}$ at $\boldsymbol{x}$ to be $\boldsymbol{0}$ just as expected. As a result, $\boldsymbol{v}$ will be an all-zero field in the interior of the solid phase, leading to $E_{m,\mu} = E_{m,\lambda} = 0$ regardless of the choice of $\boldsymbol{K}_m$ and λ. We suggest $\boldsymbol{K}_m = \boldsymbol{I}$ and $\lambda = \lambda_0$ in this case, modeling the solid phase as an isotropic,

quasi-incompressible material that impedes fluid.

No-slip-boundary material It remains to show how a proper combination of $\boldsymbol{K}_m$, $\boldsymbol{K}_f$, and λ can represent solid-fluid boundary conditions inside the domain $\mathcal{B}$. We consider two types of solid-fluid boundaries in this work: no-slip and free-slip. Note that a no-slip solid-fluid boundary simply states the flow velocity near the boundary should be zero, so we can treat it the same way as we define the solid material above, i.e., $\boldsymbol{K}_f = k_f \boldsymbol{I}$ with $k_f \to +\infty$, $\boldsymbol{K}_m = \boldsymbol{I}$, and $\lambda = \lambda_0$.

Free-slip-boundary material The last and most challenging case is to model free-slip boundary conditions (Eqn. (3.3)) with a proper choice of ($\boldsymbol{K}_m$, $\boldsymbol{K}_f$, λ). For brevity, we will present our results in 3D only. Consider a point $\boldsymbol{x}$ on a free-slip solid-fluid boundary, and let $\boldsymbol{n}$ be its unit normal vector. We augment $\boldsymbol{n}$ with two unit vectors $\boldsymbol{t}_1, \boldsymbol{t}_2$ orthogonal to $\boldsymbol{n}$ so that the matrix $\boldsymbol{R} := (\boldsymbol{n}, \boldsymbol{t}_1, \boldsymbol{t}_2)$ defines an orthonormal basis in $\mathcal{R}^3$.

To derive a proper $\boldsymbol{K}_m$ for free-slip boundaries, we recall that the isotropic energy $\|\nabla \boldsymbol{v}\|_F^2$ in the original Stokes flow model is the variational form of $\Delta \boldsymbol{v}$, the Laplacian of the velocity field component-by-component, in the PDE form of Stokes flow [5, 6]. Intuitively, this states that Stokes flow creates a component-by-component as-harmonic-as-possible velocity field, subject to the (quasi-)incompressibility constraint.

We further point out that both the Frobenius norm and the Laplacian are rotationally invariant, allowing us to change the coordinate systems freely when computing $\nabla \boldsymbol{v}$. Therefore, we can consider computing $\nabla \boldsymbol{v}$ in a local frame spanned by the normal and tangent directions at $\boldsymbol{x}$, i.e., the columns of $\boldsymbol{R}$:

$$\boldsymbol{v}_{\boldsymbol{R}} := \boldsymbol{R}^\top \boldsymbol{v}, \tag{3.9}$$

$$\boldsymbol{x}_{\boldsymbol{R}} := \boldsymbol{R}^\top \boldsymbol{x}, \tag{3.10}$$

$$\nabla_{\boldsymbol{x}_{\boldsymbol{R}}} \boldsymbol{v}_{\boldsymbol{R}} = \boldsymbol{R}^\top \nabla \boldsymbol{v} \boldsymbol{R}. \tag{3.11}$$

Here, the subscript $(\cdot)_{\boldsymbol{R}}$ means the quantity is defined in the local frame spanned by $\boldsymbol{R}$. For example, the first row in $\nabla_{\boldsymbol{x}_{\boldsymbol{R}}} \boldsymbol{v}_{\boldsymbol{R}}$ represents the spatial gradient of the normal flow magnitude.

It is now straightforward to see how the physical intuition behind free-slip boundaries can motivate the definition of $\boldsymbol{K}_m$: Essentially, free-slip boundaries retain the physical property of Stokes flow along the tangent directions and dismiss any spatial gradients along the normal direction. From the perspective of the Laplacian operator, this means directly dropping the second-order derivative along the normal direction and requiring the flow to be harmonically smooth only along the tangent directions. Mapping it back to the variational form, we see it is equivalent to zeroing out the column in $\nabla_{\boldsymbol{x}_{\boldsymbol{R}}} \boldsymbol{v}_{\boldsymbol{R}}$ that corresponds to the normal direction, leading to the following energy density to be integrated in $E_{m,\mu}$:

$$\Psi_{m,\mu} := \mu \|\nabla_{\boldsymbol{x}_{\boldsymbol{R}}} \boldsymbol{v}_{\boldsymbol{R}} \boldsymbol{\Lambda}(0,1,1)\|_F^2 \tag{3.12}$$

$$= \mu \|\boldsymbol{R}^\top (\nabla \boldsymbol{v}) \boldsymbol{R} \boldsymbol{\Lambda}(0,1,1)\|_F^2 \tag{3.13}$$

$$= \mu \|(\nabla \boldsymbol{v})(\boldsymbol{0}, \boldsymbol{t}_1, \boldsymbol{t}_2)\|_F^2 \tag{3.14}$$

$$= \mu \mathrm{trace}((\nabla \boldsymbol{v})(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^\top)(\nabla \boldsymbol{v})^\top) \tag{3.15}$$

$$= \mu \|\nabla \boldsymbol{v} (\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^\top)^{\frac{1}{2}}\|_F^2 \tag{3.16}$$

where $\boldsymbol{\Lambda}$ constructs a diagonal matrix from its input. Comparing Eqn. (3.16) with Eqn. (3.6), we see $\boldsymbol{K}_m$ should be defined as follows:

$$\boldsymbol{K}_m = \boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^\top. \tag{3.17}$$

Similarly, we propose the following $\boldsymbol{K}_f$ to impede the normal flow in E_f:

$$\boldsymbol{K}_f = \boldsymbol{R}\boldsymbol{\Lambda}(k_f, 0, 0)\boldsymbol{R}^\top = k_f \boldsymbol{n}\boldsymbol{n}^\top, \tag{3.18}$$

where $k_f \to +\infty$. Plugging this definition into E_f in Eqn. (3.8) will confirm that the proposed $\boldsymbol{K}_f$ leads to the expected behavior: It first converts $\boldsymbol{v}$ to a local frame spanned by $\boldsymbol{R}$ then forces the normal component of the flow to be zero and keeps the tangent flow intact.

Finally, regarding the choice of λ, we set its value to 0 so that $E_{m,\lambda}$ is ignored for points on free-slip boundaries. We justify this decision from two perspectives: First, using a positive λ in this case will attempt to preserve fluidic volume at free-slip boundaries, which typically means enforcing zero divergence in a neighborhood near the boundary after discretization. For example, it may suggest that a mixed cell's solid and fluid velocities be divergence-free even though they come from different phases, implying a physically problematic constraint imposed on the discretized problem. Such constraints typically lead to conditioning issues in the numerical system after discretization that calls for a more careful, subcell-accurate treatment of these mixed cells. Second, and more importantly, the real objective that incompressibility plays near the boundary is to avoid fluid leakage out of the boundary, which is automatically satisfied as long as we disallow normal flows crossing the boundary and ensure the interior of the fluid phase is properly incompressible. Both reasons imply that it is unnecessary to consider a positive λ that encourages the (ill-defined) divergence at locations on free-slip boundaries to be zero, hence our decision to set $\lambda = 0$.

Summary To conclude, we have presented an anisotropic material model which provides a uniform representation of different phases and boundary conditions encountered in the quasi-incompressible Stokes flow problem. We summarize the anisotropic material parameters $\boldsymbol{K}_m$, $\boldsymbol{K}_f$, and λ for all cases in Table 3.1.

Table 3.1: We summarize the anisotropic material parameters for modeling different phases and boundary conditions in quasi-incompressible Stokes problem: $\lambda_0 \in \mathcal{R}^+$ represents a predefined scalar parameter controlling the incompressibility in the quasi-incompressible Stokes flow; $k_f \in \mathcal{R}^+$ is a scalar parameter that determines the material's impedance for fluid flow, with $k_f \to +\infty$ creating the solid phase; $\boldsymbol{n} \in \mathcal{R}^d$ is the unit normal vector on the solid-fluid boundary that decides the material's anisotropic responses to normal and tangent flows.

	Fluid	Solid	No-slip boundary	Free-slip boundary
$\boldsymbol{K}_m$	$\boldsymbol{I}$	$\boldsymbol{I}$	$\boldsymbol{I}$	$\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^\top$
$\boldsymbol{K}_f$	$\boldsymbol{0}$	$k_f\boldsymbol{I}$	$k_f\boldsymbol{I}$	$k_f\boldsymbol{n}\boldsymbol{n}^\top$
λ	λ_0	λ_0	λ_0	0

3.4 Numerical Simulation

This section outlines the discretization scheme and the numerical solver for the continuous Stokes flow model described in the previous section. The Stokes flow system, described in Section 3.3, is discretely modeled over the entire domain and parameterized to represent fluid, solid and fluid-solid interface regions on the domain. Our discretization enables easy specification of boundary conditions, allows seamless application of additional design constraints over the domain, such as volume fraction, and accommodates a highly flexible parameter space that helps optimize for complex designs for fluid-solid interfaces. We use a uniform Cartesian lattice to discretize domain descriptors and state variables. Fluid velocities $\boldsymbol{v}$ are stored at grid nodes and bilinearly (2D) or trilinearly (3D) interpolated, while the design parameters $\boldsymbol{K}_m, \boldsymbol{K}_f$ and λ are treated as having a constant value on each grid cell (as we will see, they are stored indirectly, through some other parameters that are ultimately stored per-cell). We design a solver that solves for flow for a given parameter set, using a symmetric positive-definite (SPD) stiffness matrix. The solver also includes additional constraints, such as block divergence, to satisfy the flow divergence constraint in an aggregate fashion.

3.4.1 Anisotropic Material Parametrization

Our anisotropic material is characterized by SPD matrix fields $\boldsymbol{K}_m$ and $\boldsymbol{K}_f$ and a scalar field λ in the whole domain. From Table 3.1, we notice that these fields can be induced from the solid/fluid material distributions and boundary normals, which have lower degrees of freedom than the full SPD matrices. This inspires us to reparametrize $\boldsymbol{K}_m$, $\boldsymbol{K}_f$, and λ with three new fields: the fluidity $\rho : \mathcal{B} \to [0, 1]$, which assigns 0 to pure solid and 1 to pure fluid; the isotropy $\epsilon : \mathcal{B} \to [0, 1]$ in which larger ϵ means more isotropic material; the anisotropic orientation $\boldsymbol{\alpha} : \mathcal{B} \to \mathcal{R}^{d-1}$, which is a field of rotational angles in 2D and spherical coordinates in 3D. Furthermore, we compute a unit normal field $\boldsymbol{n} : \mathcal{B} \to \mathcal{R}^d$ induced from $\boldsymbol{\alpha}$.

Constructing $\boldsymbol{K}_m$ We define the material matrix field $\boldsymbol{K}_m$ as follows:

$$\boldsymbol{K}_m = \boldsymbol{I} - (1 - \epsilon)\rho \boldsymbol{n}\boldsymbol{n}^\top. \tag{3.19}$$

Therefore, $\boldsymbol{K}_m \approx \boldsymbol{I}$ whenever $\epsilon \approx 1$ (isotropic material) or $\rho \approx 0$ (solid phase), and $\boldsymbol{K}_m$ becomes highly anisotropic only if $\epsilon \approx 0$ and $\rho \approx 1$, i.e., anisotropic fluid that we use to represent free-slip boundaries.

Constructing $\boldsymbol{K}_f$ We define k_f with the following nonlinear mapping function borrowed from previous work [6]:

$$k_f(\rho) = k_{f_{\max}} + (k_{f_{\min}} - k_{f_{\max}})\rho \frac{1 + q}{\rho + q}, \tag{3.20}$$

where $k_{f_{\min}} \approx 0$ and $k_{f_{\max}} = 1e5$ indicate the range of k_f and $q = 0.1$ is a hyperparameter that controls the sharpness of the mapping: A smaller q generates a more binary mapping with the output k_f concentrating on the bound values. Intuitively, $k_f(\rho)$ is a monotonically

decreasing function that maps small ρ (solid phase) to $k_{f_{\max}}$ and large ρ (fluid phase) to $k_{f_{\min}}$. We then define $\boldsymbol{K}_f$ as follows:

$$\boldsymbol{K}_f = k_f(\rho)\boldsymbol{I} + (k_f(\epsilon\rho) - k_f(\rho))\boldsymbol{n}\boldsymbol{n}^\top. \tag{3.21}$$

We can see such a definition matches what Table 3.1 suggests for each type of material: For fluid phase, which satisfies $\epsilon \approx \rho \approx 1$, we have $k_f(\epsilon\rho) \approx k_f(\rho) \approx 0$, and therefore $\boldsymbol{K}_f \approx \boldsymbol{0}$; for solid phases and no-slip boundaries, we have $\epsilon \approx 1$ but $\rho \approx 0$, which leads to a large $k_f(\rho)$ and $\boldsymbol{K}_f \approx k_{f_{\max}}\boldsymbol{I}$; finally, for free-slip boundaries, we model them as anisotropic fluid with $\epsilon \approx 0$ and $\rho \approx 1$, which means $k_f(\epsilon\rho) \gg k_f(\rho) \approx 0$ in the definition, and it follows that $\boldsymbol{K}_f \approx k_{f_{\max}}\boldsymbol{n}\boldsymbol{n}^\top$.

Constructing λ Finally, we construct the λ field as follows:

$$\lambda = \lambda_{\min} + [1 - (1 - \epsilon)\rho]^p \lambda_{\max}, \tag{3.22}$$

where $\lambda_{\min} = 0.1$ and $\lambda_{\max} = 1e3$ sets the range of λ and $p = 12$ is a power index that pushes λ to be a binary choice between $\lambda_{\min}$ and $\lambda_{\min} + \lambda_{\max}$. Similar to $\boldsymbol{K}_m$ defined above, this definition of λ requires that $\lambda \approx \lambda_{\min}$ only if $\epsilon \approx 0$ and $\rho \approx 1$, i.e., near free-slip boundaries.

Summary In this manner, the anisotropic material distribution in the entire domain is parameterized by three fields ρ, ϵ, and $\boldsymbol{\alpha}$. Comparing the results above with Table 3.1, we can see that these new fields implement the intended properties of the anisotropic material in fluid, solid, and flexible solid-fluid boundaries.

3.4.2 Discretization Scheme

We now describe our discretization scheme for solving the Stokes flow problem in Sec. 3.3. We solve the energy minimization problem defined by Eqns. (3.4-3.5) on a regular grid of N^d cells and store all physical quantities at the center of each cell except for flow velocities, which we store at grid nodes. We first describe how we discretize the material parameters. Next, we discretize the variational form and introduce block divergence, a novel numerical treatment of the (quasi-)incompressibility constraint that improves the condition number of the numerical system to be solved.

Discretizing material parameters We discretize the continuous fields ρ, ϵ, and $\boldsymbol{\alpha}$ on the regular grid by storing their values at each cell center. For brevity, we use $\boldsymbol{\theta}$ from now on to refer to the set of discretized material parameters ρ, ϵ, and $\boldsymbol{\alpha}$:

$$\boldsymbol{\theta} := \{\rho, \epsilon, \boldsymbol{\alpha}\}. \tag{3.23}$$

The induced fields $\boldsymbol{K}_m$, $\boldsymbol{K}_f$, and λ are also computed and stored at these cell centers based on the mapping described previously. These parameters are treated as constant throughout the spatial extent of each cell; quadrature rules that are discussed next that need to access such parameters at quadrature point locations throughout the cell will just use the same constant value assigned to such cell.

Discretizing fluid velocity We discretize and store the fluid velocity field $\boldsymbol{v}$ on the nodes of each cell in the grid. Then, we compute the velocity and its spatial gradients at any point with bi/trilinear interpolation of the nodal values.

Discretizing variational energy terms With the material parameters and fluid velocity fully discretized, we can now discretize the variational form in Eqns. (3.4-3.5). Here, we will strategically use a different numerical integration scheme for each of the energy terms in Eqns. (3.6-3.8) as detailed next:

For the term $E_{m,\mu}$ in Eqn. (3.6) – we can label this the *Anisotropic Laplace* term, recognizing that the isotropic case $\boldsymbol{K}_m = \mathbf{I}$ would yield the Laplace term in the PDE version of Stokes, as in Du et al. [5] – we use the Gaussian-Legendre quadrature rule to integrate this energy numerically within each grid cell. Hence, we employ four quadrature points in 2D and eight quadrature points in 3D, in a fashion directly analogous to prior work [5]. As mentioned, we use the single value of $\boldsymbol{K}_m$ associated with the cell at all quadrature points.

The term $E_{m,\lambda}$ in Eqn. (3.7) can be labeled the *cell-wise divergence term*, and essentially seeks to enforce volume preservation at each individual cell where it is applied with $\lambda \neq 0$. For a given grid cell $\mathcal{C}$, we approximate this term with the following expression:

$$E^{\mathcal{C}}_{m,\lambda} \approx \lambda_{\mathcal{C}} W_{\mathcal{C}} \left[\frac{1}{W_{\mathcal{C}}} \int_{\mathcal{C}} (\nabla \cdot \boldsymbol{v}) dx \right]^2$$

where $W_{\mathcal{C}}$ is the volume (h^2 in 2D, h^3 in 3D) of the cell $\mathcal{C}$. Intuitively, what this approximation suggests is that instead of penalizing a non-zero divergence at each individual interior point of the cell, we only seek to drive the *net flux* $\text{Flux}(\mathcal{C}) = \int_{\mathcal{C}} (\nabla \cdot \boldsymbol{v}) dx$ to zero, that treats the entire cell in an aggregate fashion (allowing a non-zero divergence at interior locations, as long as the aggregate net flux is zero). This is an important modification to the numerical discretization that circumvents locking behaviors for highly incompressible materials, and has been shown effective and compatible with mixed FEM formulations for highly incompressible materials [81]. What is more, since the integrand is linear in the nodal velocities (when using bilinear/trilinear interpolation), it is possible to obtain an analytic expression for the flux, namely in 2D (if $\boldsymbol{v} = (u, v)$ are the individual scalar velocity components):

$$\text{Flux}(\mathcal{C}) = h \left(\frac{u_{10} + u_{11}}{2} + \frac{v_{01} + v_{11}}{2} - \frac{u_{00} + u_{01}}{2} - \frac{v_{00} + v_{10}}{2} \right) \tag{3.24}$$

where the subscripts denote each of the four cell vertices. The four terms of this expression can be easily and intuitively identified as, respectively, the (average) signed fluxes through the right, top, left, and bottom faces of the cell. We would exactly arrive at this same analytic expression if we simply used a 4-point Gauss quadrature for the flux integral, since the linear integrand would be integrated exactly. An exactly analogous expression can be derived in 3D; the only difference is that we would create the average velocity of each face by averaging four nodal velocities, and the scale factor of h^2 would replace h in Eqn. (3.24) to account for the area of each 3D cell.

Finally, for the term E_f in Eqn. (3.8) we employ a quadrature scheme that uses the cell vertices themselves as quadrature points, namely:

$$E^{\mathcal{C}}_f \approx \frac{W_{\mathcal{C}}}{2^d} \sum_I \| \boldsymbol{K}_f^{\frac{1}{2}}(\mathcal{C}) \boldsymbol{v}_I \|_2^2$$

where the index I traverses all cell vertices (4 in 2D; 8 in 3D). We observe that this term seeks to model a cell as viscous/rigid (in the solid phase) or permeable (in the fluid phase), hence it is perfectly reasonable to apply this viscous penalty on a per-vertex basis. Doing so, in fact, avoids certain hazards of low-order quadrature schemes (for example a single-point quadrature scheme evaluated at the cell center could yield a zero result even if non-zero velocities at the vertices happen to average out to zero at the cell center).

After combining all terms in this discretization scheme, we arrive at a quadratic energy form, given by $E = \boldsymbol{v}^\top \boldsymbol{K} \boldsymbol{v} - \boldsymbol{b}^\top \boldsymbol{v}$, where $\boldsymbol{v}$ stacks all velocity degrees of freedom and $\boldsymbol{K}$ and $\boldsymbol{b}$ the SPD matrix and vector composed of the material parameters, respectively. We enforce the Dirichlet boundary condition by forcing the corresponding nodal velocities to be the desired value. Putting them together, we state the discrete variational form as the following quadratic programming problem:

$$\min_{\boldsymbol{v}} \quad \boldsymbol{v}^\top \boldsymbol{K}(\boldsymbol{\theta}) \boldsymbol{v} - \boldsymbol{b}(\boldsymbol{\theta})^\top \boldsymbol{v} \tag{3.25}$$

$$s.t. \quad \boldsymbol{v}_i = (\boldsymbol{v}_D)_i, \forall (i, (\boldsymbol{v}_D)_i) \in \mathcal{D}, \tag{3.26}$$

where $\mathcal{D}$ states the Dirichlet boundary conditions.

3.4.3 Block divergence

The variational form and the anisotropic material parametrization frequently use large hyper-parameters to satisfy constraints that are supposed to be strict, e.g., perfect incompressibility is approximated with $E_{m,\lambda}$ penalizing the divergence with a coefficient $\lambda_{\max} \to +\infty$, and free-slip boundaries are replaced with E_f scaled by $(k_f)_{\max} \to +\infty$ that penalizes nonzero normal flows. Unfortunately, while setting such parameters to near infinity tightens these constraints from an optimization perspective, it also leads undesirable behaviors. Specifically, using an exceptionally high λ value is known to compromise the conditioning of the system. Using a high k_f value has an even more obscure side effect; since boundary cells use this penalty to prevent flows from having a component in the *normal* direction to the boundary, if there are any two neighboring boundary cells that have even slightly different directions of anisotropy, a high k_f value would effectively cause the velocities at any shared vertices between the two cells to be driven to zero, as their projection to two non-parallel directions (the normals at the two cells) will jointly and strongly be required to be zero. Hence, this might inadvertently once again drive us to a situation where we are unintentionally forcing a no-slip condition.

Using moderately-high values for λ and k_f certainly helps alleviate these issues. The risk of doing so, however, is that we open up the possibility for volume loss that, even not egregious at the local level, could add up to a substantial flow loss at the global scale. This is aggravated by the fact that (with sound motivation) we do not enforce incompressibility (we use $\lambda = 0$) at boundary cells paired with a free-slip condition, depending on the zero-normal-flow condition (which is not strictly enforced if k_f is not very high) to prevent leakage at free-slip boundaries.

We introduce an original solution to this issue, by pairing the moderately-high parameters at the per-cell level with a hard-constraint of absolute volume preservation at a more aggregate scale, namely large blocks (typically rectangular boxes of 4 or 8 cells across) that we partition

our domain in. This is illustrated in Figure 3.11 (right); we refer to the associated ablation study for an illustration of the effect of this technique (or the consequences of its omission).

We partition the domain $\mathcal{B}$ into large blocks (indexed by b) of uniform sizes and enforce the (aggregate) net flux over each block to be zero. Since the per-cell flux is a linear expression on the nodal velocities, the aggregate constraint is simply the sum:

$$0 = \text{Flux}(\mathcal{B}_b) = \sum_{\mathcal{C} \in \mathcal{B}_b} \text{Flux}(\mathcal{C})$$

As one would intuitively expect, the net flux over the block $\mathcal{B}_b$ is a linear expression of the averaged (signed) fluxes through the faces of the aggregate block; the contribution of faces interior to $\mathcal{B}_b$ cancels out when fluxes of neighboring cells are summed. Ultimately, the net flux constraint on each block yields a single linear equation (with a zero right-hand side), and these *block-divergence constraints* for the entire domain are ultimately distilled into a linear constraint system $\mathcal{C}\boldsymbol{v} = \mathbf{0}$. Ultimately, we reformulate our governing anisotropic Stokes equations as a linearly-constrained, quadratic optimization problem, where we minimize the functional $E(\boldsymbol{v}) = \boldsymbol{v}^T \mathbf{K}(\boldsymbol{\theta})\boldsymbol{v} - \boldsymbol{b}(\boldsymbol{\theta})^T \boldsymbol{v}$ (resulting from the quadrature schemes in the previous paragraph), subject to the linear constraint $\mathcal{C}\boldsymbol{v} = \mathbf{0}$; $\boldsymbol{\theta}$ represents the design parameters. We ultimately solve this constrained optimization problem via the Karush-Kuhn-Tucker condition, in the system:

$$\begin{pmatrix} 2\mathbf{K}(\boldsymbol{\theta}) & \mathbf{C}^T \\ \mathbf{C} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{v} \\ \mathbf{q} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}(\boldsymbol{\theta}) \\ 0 \end{pmatrix}$$

(where $\mathbf{q}$ are the Langrange multipliers associated with the block-divergence constraint). We use direct factorization methods (we employ PARDISO [82]) to solve these symmetric/indefinite problems both for forward simulation as well as for the inverse problems associated with optimization; we may omit, for brevity, an explicit reference to the constraint in our upcoming discussion of the optimization pipeline, with the understanding that it is always present in our pipeline. From a practical perspective, we found this technique to be highly effective; in our 3D examples, enforcing a hard block-divergence constraint on 4x4x4 blocks, paired with moderate to moderately-high parameters λ and k_f produced results that were practically indistinguishable from using a very high λ, and much more resilient to artifacts associated with using high k_f values. Note that the block-divergence constraint does not depend on the design parameters $\boldsymbol{\theta}$, contrasted to $\mathbf{K}$ and $\boldsymbol{b}$ that do. Ultimately, we view the solution of this constraint optimization problem as the *simulation* function $\boldsymbol{v} = F(\boldsymbol{\theta})$ that maps the design parameters to the corresponding simulation result.

3.5 Optimization

We now describe our optimization problem which is built upon the numerical simulator described before. We first formulate the fluidic device design task as a numerical optimization problem and state its formal definition. Next, we describe the algorithm for solving this optimization problem numerically.

3.5.1 Problem Statement

We formally define the task of designing a fluidic device as the following numerical optimization problem:

$$\min_{\boldsymbol{\theta}} \quad L_f(\boldsymbol{v}) + w_c L_c(\boldsymbol{\theta}) + w_d L_d(\boldsymbol{\theta}) + w_a L_a(\boldsymbol{\theta}), \tag{3.27}$$

$$s.t. \quad \boldsymbol{v} = F(\boldsymbol{\theta}), \tag{3.28}$$

$$\boldsymbol{\theta}_{\min} \leq \boldsymbol{\theta} \leq \boldsymbol{\theta}_{\max}, \tag{3.29}$$

$$0 \leq V_{\text{iso-fluid}}(\boldsymbol{\theta}) \leq V_{\max}, \tag{3.30}$$

$$0 \leq V_{\text{all-fluid}}(\boldsymbol{\theta}) \leq V_{\text{b}} + V_{\max}. \tag{3.31}$$

Here, L_f states the *functional* loss given by the task specification, which is typically defined as the L_2 difference between the outlet flow in simulation and the desired outlet flow profile.

Regularizers The next three terms in the objective are regularizers on the material parameter $\boldsymbol{\theta}$: Following the standard practice in the previous field-based fluidic topology optimization method [6], the *compliance* regularizer L_c computes the elastic energy accumulated from enforcing the outlet flow profile to be the same as the target as extra Dirichlet constraints:

$$L_c(\boldsymbol{\theta}) := \boldsymbol{v}_c^\top \boldsymbol{K}(\boldsymbol{\theta}) \boldsymbol{v}_c - \boldsymbol{b}(\boldsymbol{\theta})^\top \boldsymbol{v}_c, \tag{3.32}$$

$$\boldsymbol{v}_c = F(\boldsymbol{\theta}; \mathcal{D} \cup \mathcal{D}_O), \tag{3.33}$$

where $\mathcal{D}_O$ summarizes the extra Dirichlet conditions from the target outlet flow profile. The motivation behind L_c is that a lower elastic energy means the outlet flow profile will be more similar to the target if we release the extra Dirichlet conditions on the outlet.

Next, the *directional* regularizer L_d is defined as the differences between the normal direction of an anisotropic cell and the normals from its neighborhood. We first filter out cells that contain anisotropic materials with two thresholds ϵ_0 and ρ_0: a cell is considered to be anisotropic if its $\epsilon < \epsilon_0$ and $\rho > \rho_0$, i.e., it is modeled as anisotropic fluid. Let $\mathcal{A}$ be the set of anisotropic cells, we define L_d as follows:

$$L_d := \sum_{c \in \mathcal{A}} 1 - \psi_{\cos}(\boldsymbol{\alpha}_c, \boldsymbol{\alpha}_{\text{nbr}}) \tag{3.34}$$

where $\boldsymbol{\alpha}_c$ is the anisotropic direction (a unit vector) of cell c, $\boldsymbol{\alpha}_{\text{nbr}}$ an average direction fitted from its small neighbors ($3 \times 3 \times 3$ in our implementation), and $\psi_{\cos}$ the cosine similarity between them. Therefore, minimizing L_d encourages smooth free-slip solid-fluid boundaries.

Finally, the *anisotropic* regularizer L_a concentrates anisotropic cells near solid-fluid boundaries:

$$L_a(\epsilon) := \sum_c \epsilon_c \rho_c (\rho_{\max}^{local} - \rho_{\min}^{local}), \tag{3.35}$$

where the sum loops over each cell c and $\rho_{\max}^{local}$ and $\rho_{\min}^{local}$ are the maximum and minimum fluidity from its small neighborhood ($3 \times 3 \times 3$ in our implementation). The proposed loss encourages cells near the solid-fluid boundaries (large $\rho_{\max}^{local} - \rho_{\min}^{local}$) to use anisotropic materials (small ϵ). To avoid chasing a moving target, we freeze $\rho_{\max}^{local}$ and $\rho_{\min}^{local}$ in L_d when optimizing ρ in each iteration.

Constraints Apart from the objective function, the optimization problem also contains a number of constraints: Eqn. (3.28) ensures the flow field $\boldsymbol{v}$ is computed from the numerical simulator, and Eqn. (3.29) states the bound constraints on the material parameters. The last two equations (3.30) and (3.31) define volume constraints on the fluidic region and the free-slip fluid-solid boundaries:

$$V_{\text{iso-fluid}} := \sum_c \epsilon_c \rho_c, \tag{3.36}$$

$$V_{\text{all-fluid}} := \sum_c \rho_c. \tag{3.37}$$

The difference between them implies the volume of anisotropic cells, and $V_{\max}$ and V_b are two task-dependent thresholds defining the maximum fluidic volume and the maximum volume of anisotropic cells (free-slip boundaries), respectively.

Summary In summary, the optimization problem aims to find an optimal material distribution $\boldsymbol{\theta}$ that minimizes the functional loss under bound constraints and volume constraints. The optimization process is facilitated by three regularizers that encourage spatially smooth yet clear solid-fluid boundaries.

3.5.2 Numerical Optimizer

Standard topology optimization typically consists of hundreds of thousands of decision variables even for moderate-size problems (e.g., on a 64^3 grid), and our numerical optimization is no exception. In fact, due to the inclusion of additional anisotropic material parameters, the numerical optimization problem we formulated has a larger number of decision variables than its isotropic topology optimization counterparts. Following the standard practice in topology optimization, we use the method of moving asymptotes (MMA) [4], a widely used gradient-based optimization algorithm, to solve this large-scale optimization problem. As MMA requires gradients with respect to the material parameters $\boldsymbol{\theta}$, we extended the numerical simulation method in Sec. 3.4 to a differentiable simulator in a way similar to Du et al. [5], through which the gradients can be computed from backpropagating the loss function. To encourage more structured designs during the optimization process, we dynamically update the upper bound on the isotropy ϵ_c for each cell c based on the following heuristic:

$$(\epsilon_c)_{\max} := 1 - (\rho^{local}_{\max} - \rho^{local}_{\min}). \tag{3.38}$$

In other words, when a cell is surrounded by both solid and fluid cells, we force it to choose anisotropic materials (small upper bound on ϵ). Our empirical experience suggests that this dynamic update scheme biased the optimization process towards generating more structured fluidic device designs.

3.5.3 Choice of Parameters and Interpolation Functions

For the hyperparameter settings, we reused the value from [6] for $k_{f_{\min}}$. We chose a near-zero value for $\lambda_{\min}$. The choices of these two hyperparameters followed the convention in topology

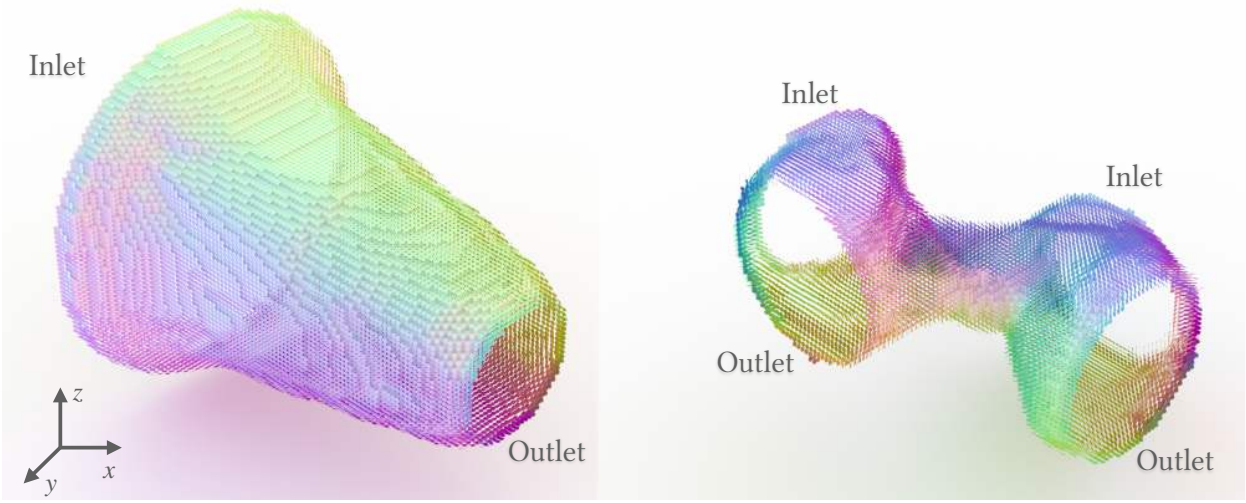


Figure 3.4: Designing an *Amplifier* (left) and a *Mixer* (right) under a grid resolution of $80 \times 80 \times 80$. The anisotropic boundaries of the optimized designs are visualized using small colored disks.

optimization. We did not observe noticeable differences when their values were perturbed. For $k_{f_{\max}}$ and $\lambda_{\max}$ values, we chose values that can balance the solver performance and block divergence (See our discussion in Sec. 3.4.3). We chose the current block size by experimenting with a minimum block size that gives satisfactory free-slip boundaries without introducing large divergence errors in small neighborhood. For the choices of interpolation functions, we used the interpolation function (Eq. 3.20) in [6] with the same q value for k_f, and we used a power-indexed interpolation function for λ, which is conventional in density-based topology optimization, to encourage the design's binary convergence. We did not observe noticeable differences between these two functions.

3.6 Applications

We demonstrate a variety of complex fluidic device designs obtained using our optimization pipeline. We use a grid resolution of $100 \times 100 \times 100$ for the Fluid Twister example and $80 \times 80 \times 80$ for all other optimization examples and initialize the material parameters to be isotropic ($\epsilon = 1$) with fluidity $\rho = V_{\max}$. We run all optimizations with 300 iterations and use the design that achieves minimum final loss as our optimized design. We report the task-specific volume fraction limit, initial and optimized functional loss, and the execution time for each task in Table 3.2.

We implement our optimization pipeline in C++ and use the implementation of PARDISO [82] for solving our linear systems and MMA [4] as our optimizer. Since the sparsity pattern of the system matrix remains the same over optimization iterations, we also optimize the matrix factorization time by performing symbolic factorization only once. We perform our experiments on an Intel Ice Lake 128-core server and Ubuntu 20.04 operating system.

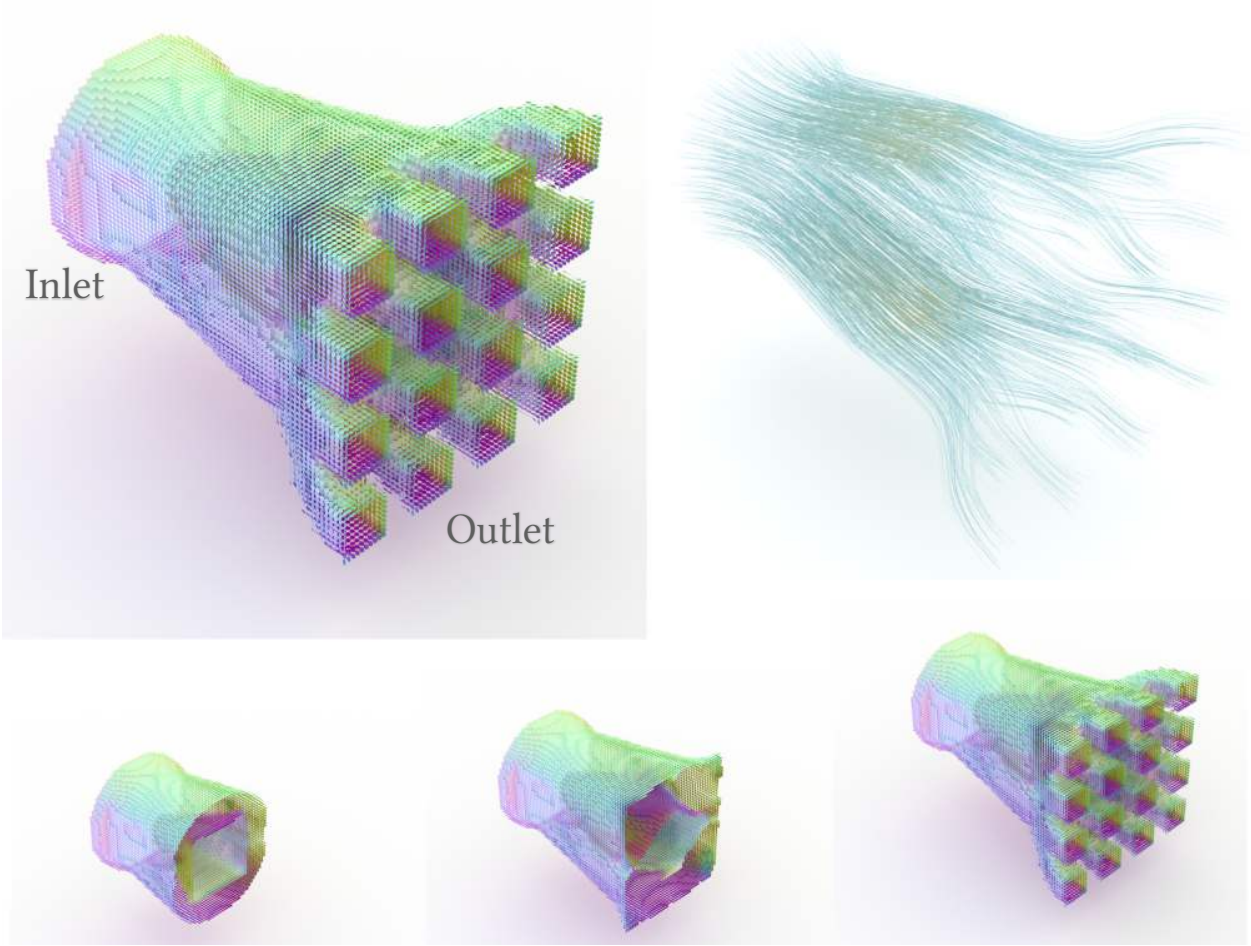


Figure 3.5: Our pipeline generates a tree diffuser on a $80 \times 80 \times 80$ grid. Top left: The anisotropic boundary of the optimized design is visualized using small colored disks. Bottom: The 3 images in the middle visualize the perpendicular cross-sections of the optimized design at different depths from the inlet. These cross-sections highlight the progressive branching from a single inlet to multiple outlets. Top right: The fluid flow, simulated from the optimized design, is visualized as streamlines.

Motivating examples As motivating examples, we present the 3D design problems of an amplifier and a mixer under a $80 \times 80 \times 80$ grid. The 2D versions of both examples are presented in prior works [5, 6]. In *Amplifier*, we enforce a constant circular input with inflow velocity $(v_{in}, 0, 0)$. The objective is to amplify the flow by a factor of $\frac{5}{3}$. In *Mixer*, the design objective is to mix a high-pressure and low-pressure flow from two inlets to produce equal middle-pressure flows at the two target outlets. Specifically, the two inlets have inflow velocities $(v, 0, 0)$ and $(0, 2v, 0)$ and the two outlets have outflow velocities $(1.5v, 0, 0)$ and $(0, 1.5v, 0)$, respectively. The volume fraction limit is 0.3 for *Amplifier* and 0.4 for *Mixer*. We visualize the optimized designs in Fig. 3.4.

Tree Diffuser The goal of this example is to generate a fluidic diffuser that directs fluids from one constant circular-shaped inlet to 16 small square-shaped outlets while bypassing

	Resolution	$V_{\max}$	L_f		Iteration time (s)			
			Initial	Final	Forward	Backprop.	MMA	Total
Twister	100×100×100	0.30	22.575	0.519	1374.1	152.6	153.2	1679.9
Tree Diffuser	80×80×80	0.25	3.966	0.133	721.9	85.0	61.1	868.0
Circuit-1	80×80×80	0.25	86.281	2.125	737.2	86.5	83.1	896.7
Circuit-2	80×80×80	0.50	86.153	1.490	721.3	86.4	82.7	889.7

Table 3.2: We report the statistics from optimizing the design problems in Sec. 3.6, including the maximum fluidic volume fraction, the functional loss before and after optimization, and the time decomposition in each optimization iteration: "Forward" represents the time spent on computing the loss function, which is dominated by the numerical simulator; "Backprop." stands for the time spent on computing the gradients; "MMA optimizer" reports the time cost from running one iteration of MMA after obtaining the loss and gradients; "Total" is the sum of all the time above.

a small obstacle at the center of the domain, which we enforce as zero-velocity Dirichlet constraints. In the optimized design (Fig. 3.5), an interesting tree-like topology automatically emerged from our pipeline, where the fluid first branches into four chambers and then into the 16 outlets. The resulting shape produced by our pipeline exhibits an intuitive design. The branching is gradual as one moves from the inlet to the interior of the domain and the branching factor increases gradually in this direction. The cross-sections in Fig. 3.5 highlight the progress of the branching in the optimized design. This example highlights the ability of our method to synthesize an intricate structure with 16-outlets without any prior on its shape or topology.

Fluid Twister In this example, we enforce a circular-shaped constant inlet with inflow velocity $(v_{in}, 0, 0)$. The objective of the task is to generate a swirl flow in the yz-plane at the outlet of the domain. This example is solved on a $100 \times 100 \times 100$ grid with nearly 4 million decision variables, and the final design is shown in Fig. 3.1. We show the streamline visualization of the optimized design in the middle of 3.1, which successfully generates a swirl flow at the outlet. Using our topology optimization approach, a propeller-like structure automatically emerged from a constant fluidity parameter field, highlighting the ability of our pipeline to create a new topology.

Fluid Circuit In this example, we mimic a fluid circuit that connects multiple inlets, located at two faces of the cubic domain, to multiple outlets located at the remaining four faces of the cubic domain. The inlets have three types of inlet velocities, and the goal of the circuit is to connect the inlets to produce equal flows at the outlets. The result of our optimization is shown in Fig. 3.6. The optimized design that emerges from our pipeline is intuitive, as it seems to connect the nearest pairs of inlets and outlets that can produce equal flows in order to meet the small volume fraction constraint (0.25 in this example). For instance, the top-right inlet on the left face (of the domain) is connected to the nearest outlet on the top face (of the domain). We present two results using different volume constraints $V_{\max} = 0.25$ and $V_{\max} = 0.5$ and observe different topological structures, which exhibit

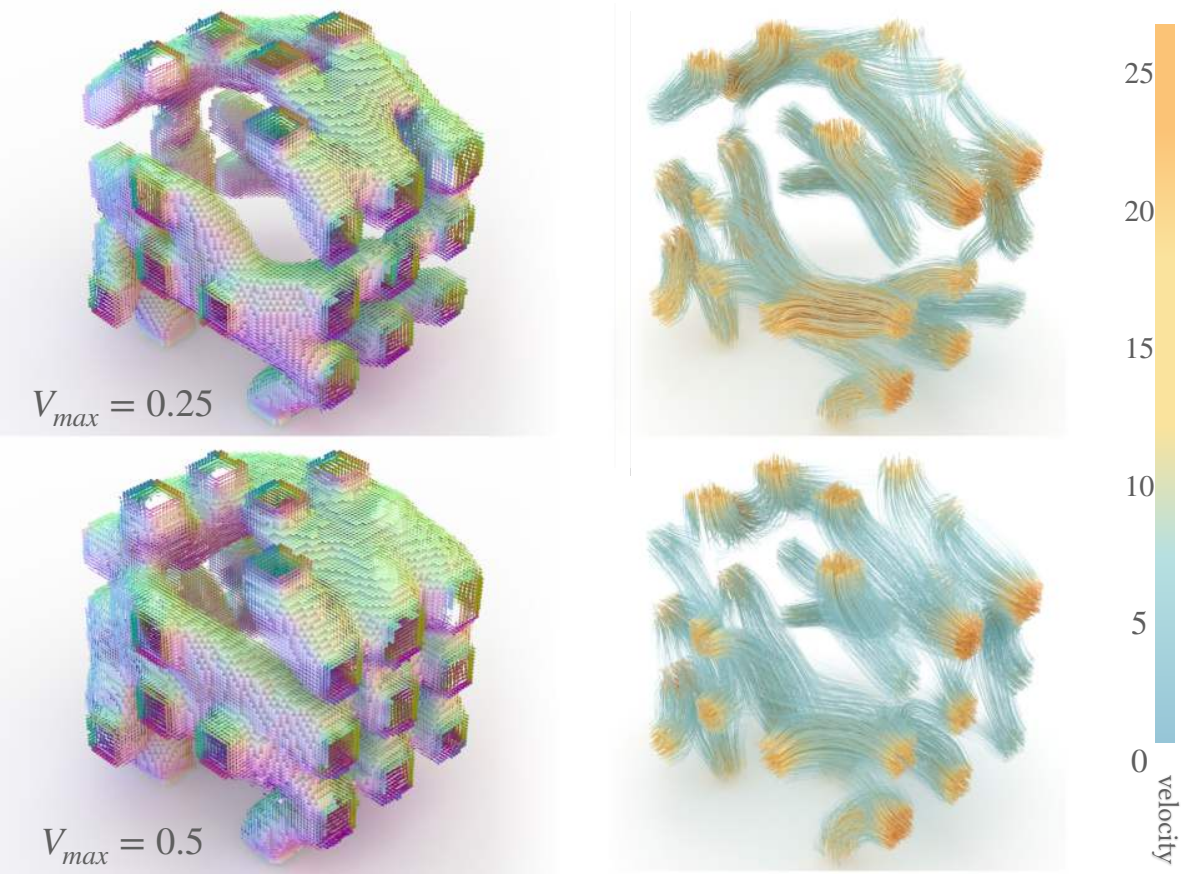


Figure 3.6: We design the *Circuit* with maximum volume fraction $V_{\max} = 0.25$ (top) and $V_{\max} = 0.5$ (bottom). Inset: We visualize the domain setup. Different faces of the domain are marked as inlet and outlet with different flow velocity. Left: The anisotropic boundary of the generated design is visualized using small colored discs. Right: The resulting fluid flow from the optimized design is visualized using colorcoded streamlines.

different routing plans between the inlets and outlets.

3.7 Evaluations

Below we show experiments evaluating our method and comparing our optimization methods to previous works [5, 6]. We include an additional evaluation of the sensitivity of our results to initialization, an experiment validating our optimized designs, and an ablation study on our new anisotropic material model.

Solver evaluation To verify that our simulation results converge under refinement, we evaluate our solver on a 2D fluid amplifier design represented by two symmetric Bézier curves. The left of the domain has horizontal inflow. We simulate the design in square domains of dimensions 32, 64, 128, 256, 512, 1024, and 2048, and observe that the velocity fields converge to a limit (Fig. 3.7). To compare our solver with more traditional Stokes flow solvers, we

simulate the same amplifier design using the solver from Du et al. [5], which is an exact interface solver that supports free-slip boundaries. As shown in Fig. 3.7, the main body of the velocity fields are similar, and the discrepancy mainly exhibits near the solid boundary due to the different boundary treatments. Specifically, Du et al. [5] assumes an exact interface for the solid-fluid interface and simulates the cells of the interface with subcell precision, while our method only assumes the direction of the interface within the cell.

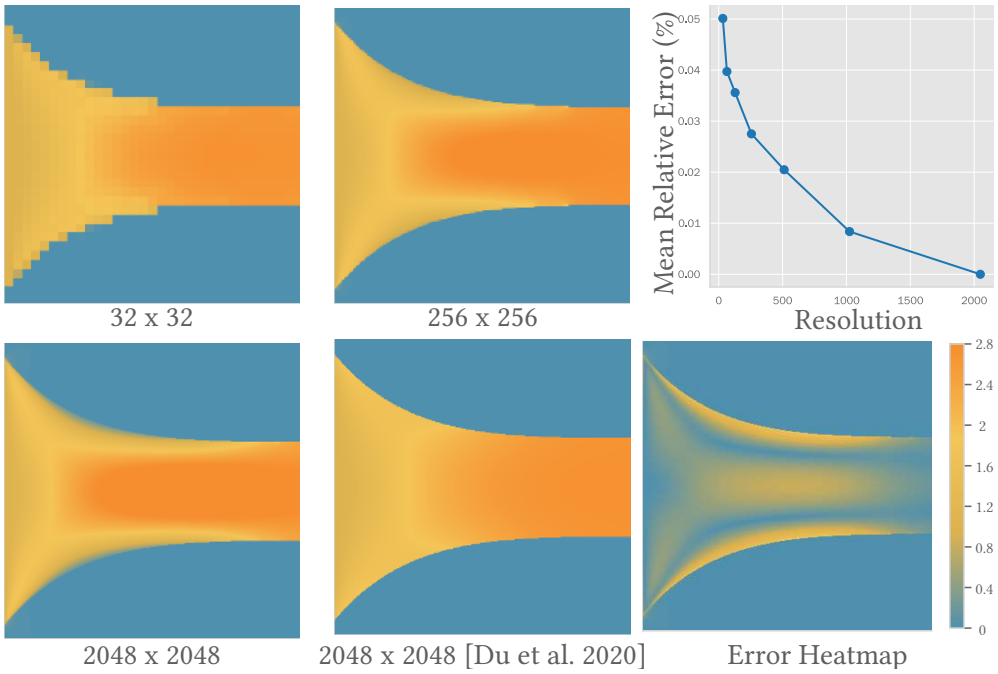


Figure 3.7: Evaluation of our solver on a 2D fluid amplifier design. We visualize the norm of the nodal velocity field simulated in square domains of dimension 32, 256 and 2048 (top left, top middle, bottom left). The relative error (to the solution at dimension 2048) monotonically decreases as the resolution increases. We additionally simulate the same design at 2048x2048 using a traditional Stokes flow solver from Du et al. [5] (bottom middle), and visualize the difference field (bottom right, normalized to inflow value).

Sensitivity of results to block size To evaluate the sensitivity of the optimization results to block size, we optimize the 2D amplifier design problem under 100×100 with volume fraction limit 0.5. The domain has an inlet with parallel horizontal inflow located at the left of the domain and an outlet at the right of the domain. We initialize the optimization with $\epsilon = 1$ and $\rho = 0.5$ and repeat the optimization with block sizes of 4, 8 and 16 (Fig. 3.9). We observe that the optimized designs are similar.

Comparison with baselines We compare our method with two representative baseline algorithms in the design and optimization of fluidic devices: One is a field-based fluidic topology optimization method described by Borrvall and Petersson [6], which uses isotropic materials with varying stiffness to model porous materials that allow or impede water passing. The other is a parametric-shape optimization algorithm [5], which uses parametric shapes to

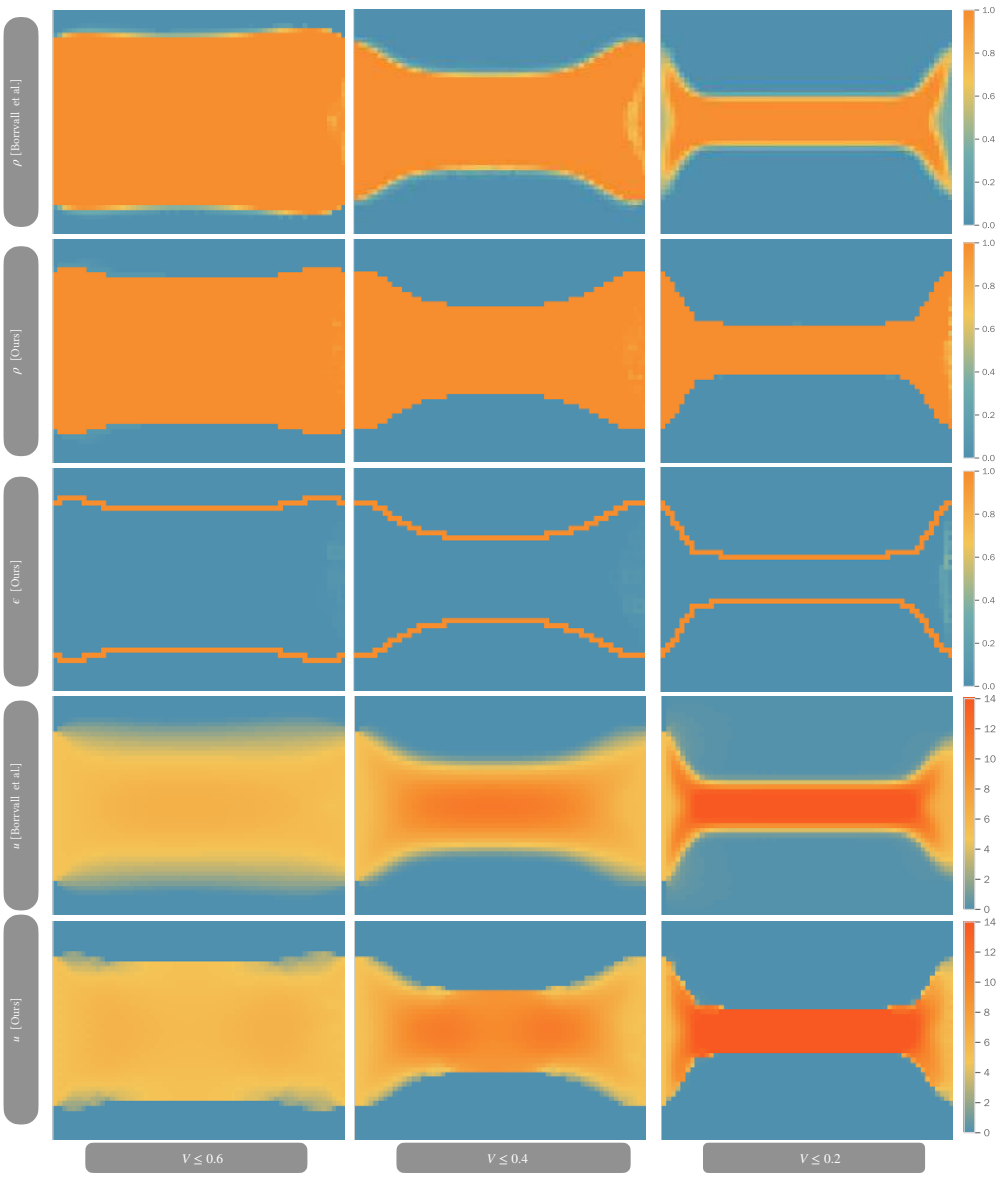


Figure 3.8: Comparisons between our method and [6] using a 2D design problem (60×60 cells). The domain has one inlet and outlet at its left and right, respectively. The goal is to synthesize a structure that transports the inlet flow with velocity $(v_i, 0)$ to the outlet. From left to right: we run both methods to solve this design problem with varying volume fraction limits from 0.6 to 0.2. For both methods, we visualize the optimized fluidity field (row 1, 2) and velocity field (row 4, 5). We additionally visualize the optimized isotropicity field ϵ for our method (row 3). As we decrease the volume fraction from left to right, the method of Borrvall and Petersson [6] starts to synthesize non-physical designs and flow, i.e., the solid cells near the inlet and outlet as well as nonzero fluid velocity on them.

represent the device boundary and optimize the design by evolving shape parameters. We implemented the method of Borrvall and Petersson [6] exactly as described in the paper with the same set of hyperparameters and verified that we can obtain the results presented. We

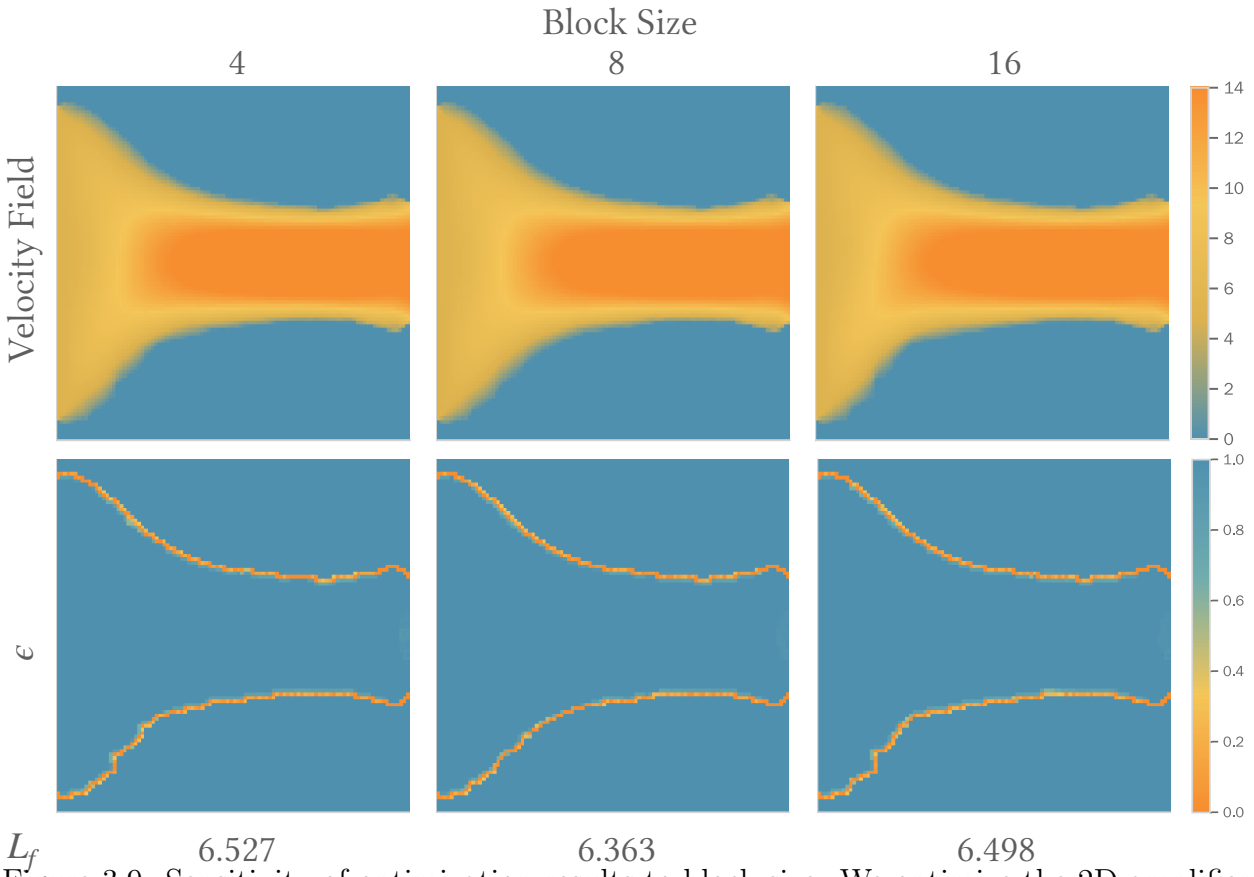


Figure 3.9: Sensitivity of optimization results to block size. We optimize the 2D amplifier design problem with block sizes 4, 8, and 16 where initial $L_f = 141.938$. We visualize the norm of the velocity field (top) and the optimized isotropy field (bottom) and report the optimized L_f value (bottom).

used the open source implementation for comparison with [5].

The main difference between our approach and [6] is the introduction of an anisotropic material model in fluidic topology optimization problems. To validate the merits of anisotropic materials, we consider a 2D design problem that aims to synthesize the internal structure between an inlet and an outlet on the opposite sides of a square domain (60×60 cells). The inlet flow is given by $(v_i, 0)$ and enforced as the Dirichlet boundaries, and the goal is to create a design so that the outlet flow at each node is $(v_i, 0)$ too. While this design problem has a trivial solution of a straight pipe connecting the inlet and the outlet, we stress test the problem by imposing various volume fraction $V_{\max}$ ranging from 60% to 20% of the domain (Fig. 3.8, left to right) and run both methods. As we decrease $V_{\max}$, the method of Borrvall and Petersson [6] starts to generate physically implausible flow velocities that co-exist with several solid cells near the inlet and the outlet (Fig. 3.8 top). The deeper reason behind them is that traditional isotropic field-based methods typically lead to blurred solid-fluid boundaries that occupy more fluid volumes than a physical boundary should. In exchange for that, such methods have to trade fluid volumes that should have stayed in the interior of the fluid phase when the volume fraction becomes tight. In contrast, our method maintains the sharp boundary of the final design and the physically plausible flow velocity even if we push

$V_{\max}$ to its minimum (Fig. 3.8 bottom), which we attribute to the anisotropic material model.



Figure 3.10: Solving the Fluidic Twister example using the open-source implementation from Du et al. [5]. Left: The parametric shape and fluid flow visualization before optimization, with the top-right insets showing the designs viewed from their outlets. Right: The corresponding visualizations after optimization.

The difference between our method and [5] is that our representation of a fluidic system is field-based while their representation is based on parametric shapes. We run both methods in the Fluidic Twister example, which was also one of the design problems studied in their paper. We report the optimization results in Figs. 3.1 and 3.10, respectively. It is evident that the design space in [5] is limited to parametric shapes that only evolve a simple surface (Fig. 3.10) without changing its topology. On the contrary, our method automatically synthesizes a propeller-like structure without geometrical or topological priors. Furthermore, our final design achieves a much lower functional loss (0.52) than their approach (3.31), indicating that we explored a much larger design space thanks to the expressiveness of the field-based representation.

Block divergence To understand the effects of the block divergence constraints in our numerical simulation, we consider a 2D example with two inlets and two outlets on the left and right sides of a square domain (Fig. 3.11). We use moderately large material parameters k_f and λ_0 defined in Table 3.1 to model the design and simulate the Stokes flow with and without block divergence. To quantify the divergence in the whole domain, we compute the ratio between the outflux at the two outlets and the influx from the two inlets, which is about 67% without block divergence and 100% with block divergence. These numbers indicate that moderate material parameters, while friendly to a numerical solver, create leaky flows disappearing into the solid phase in the domain. With block divergence, however, the whole domain remains divergence-free in an aggregated sense without messing with the conditioning of the numerical system.

3.8 Discussion and Conclusion

This chapter has several current limitations, which also suggest important directions for future work.

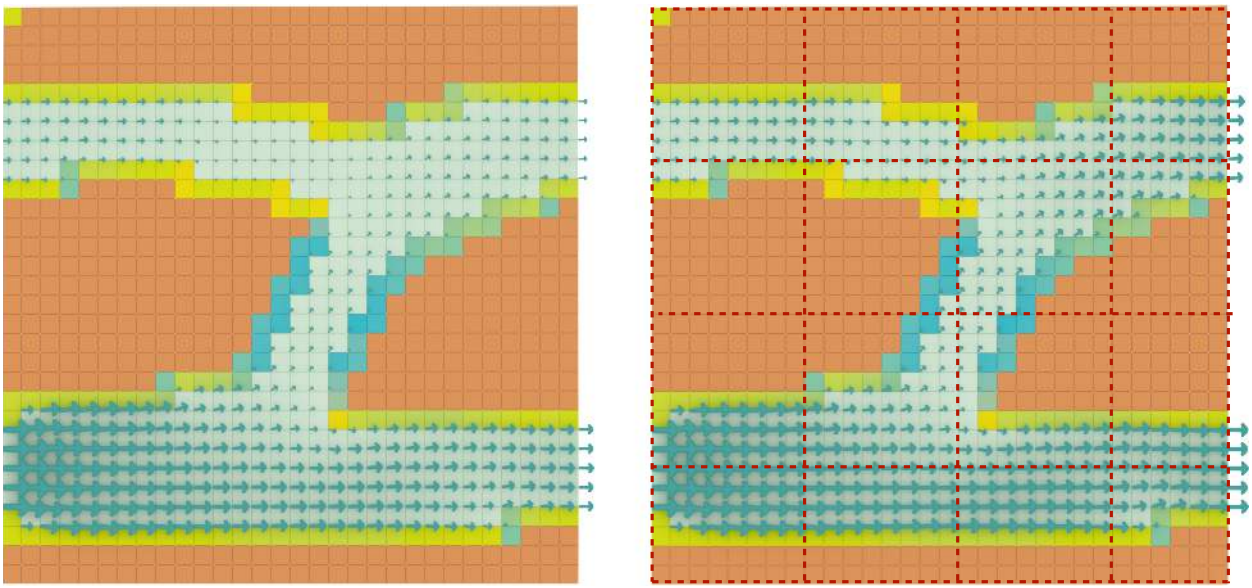


Figure 3.11: We study the effects of block divergence for simulating divergence-free Stokes flow using this two-inlet, two-outlet design problem in 2D (30×30). The inlets and outlets are defined on the left and right sides of the domain, respectively. Left: We model the design with moderate k_f and λ_0 without imposing the block divergence constraints in simulation. The outflux on the two outlets is 67% of the influx from the two inlets, indicating the flow velocities dissipate into the solid phase inside the domain. Right: We rerun the same simulation with block divergence constraints with block sizes of 8×8 (dashed red lines). The resulting outflux is 100% of the influx.

First, the present framework adopts several modeling simplifications. In particular, the fluid phase is modeled as steady-state Stokes flow and therefore does not capture time-dependent flow behavior. This assumption is shared by much of the prior work with comparable feature sets, but it limits the range of systems that can be addressed. Extending the framework to dynamic fluid systems is therefore a natural next step beyond the present Stokes-flow setting. In addition, the solid phase is modeled as rigid. Extending the current formulation to compliant solid materials would provide a foundation for optimization of coupled incompressible flow and deformable structures, which is especially relevant to the design of fluid-driven soft robotic systems.

Second, the current method remains limited in scalability. Although the demonstrated resolutions are competitive in the context of fluidic topology optimization, they remain far below the scale reached in topology optimization pipelines for purely solid or elastic systems. The present framework relies on a direct solver for the algebraic systems arising from discretization. Achieving a significant improvement in scalability will likely require iterative and possibly multiresolution solvers, such as multigrid or multigrid-preconditioned methods. While scalable multigrid solvers for Stokes flow have been studied previously [83], adapting such methods to the present setting raises several complications. For example, many prior Stokes multigrid approaches rely on staggered discretizations and mixed variational formulations for robust performance. A mixed formulation may therefore also be required here [81]. However, preserving good convergence behavior in the context of the collocated

discretization used in this chapter will require careful treatment of stability and relaxation, as well as attention to the impact of anisotropic terms on solver convergence.

Finally, beyond model fidelity and solver scalability, an important next step is to incorporate manufacturability more directly into the optimization framework. New objectives and constraints will be needed to account for fabrication requirements, and methods for manufacturing and experimentally benchmarking optimized fluidic devices remain essential for closing the gap between simulation and physical realization.

Chapter 4

Scalable Optimization and Co-Design in Dynamic Fluidic Systems

4.1 Overview

Complex fluidic systems play an important role in many engineering and scientific disciplines, encompassing applications at different length scales ranging from biomedical implants [84] and microfluidic devices [85], to hydraulic devices and flying robots [86]. Understanding these fluid-solid coupling mechanisms in nature and mimicking their control strategies in artificial designs is essential for advancing our control and design capabilities to synthesize novel solid-fluid systems.

Devising neural control algorithms to accurately manipulate the behavior of a complex fluidic system and optimize its performance remains challenging due to the intricate interplay between device geometry, control policies, flow dynamics, and the inherent physical and optimization constraints unique to each fluidic system. On one hand, differentiable fluid interactions are inherently difficult because simulation is dynamic, involving a sequence of forward and backward steps interleaved with control signals that are computationally expensive.

On the other hand, naively employing traditional control algorithms, mainly derived from their solid counterparts, to control fluidic systems remains difficult due to characterizing the infinite degrees of freedom of fluid flows and their interactions with solid boundaries. The co-design of fluid-solid systems, involving both shape and control, is critical to exploring the optimal performance of these systems.

Currently, the machine learning community lacks a computational Gym-like [87] environment to facilitate the exploration of fluidic systems manifesting strong solid-fluid interactions and controllable dynamic boundaries.

Recent literature in robotic learning (e.g., [88]) has established unified multi-physics differentiable simulation platforms to facilitate learning control policies for various fluid interactions in daily scenarios. Similar ideas can be observed in [1, 39, 89], where differentiable simulation plays a central role in accommodating various design and optimization tasks of dynamic systems involving fluid dynamics. However, despite these inspiring advances, learning the control policies and exploring the optimal performance of a dynamic fluidic

system with complex boundary conditions remains difficult due to their inherent complexities in differentiating solid boundary behaviors and optimizing their fluidic consequences due to these boundary motions.

This chapter presents a novel framework for a fully automated pipeline aimed at devising neural controls for complex fluidic systems with dynamic boundaries. Our framework is designed to robustly control complex fluidic systems that consist of externally driven soft boundaries and internal complex flow behaviors, such as those systems underpinning an artificial heart or a microfluidic device.

NeuralFluid consists of three critical components to enable neural control of a complex fluidic system. First, we devise a differentiable geometry representation to offer an expressive design space while remaining low-dimensional, enabling efficient exploration by the optimization algorithm. Second, we implement a differentiable fluid simulator with solid-fluid interface handling to accurately characterize the dynamic fluid behavior and predict its spatiotemporal impact on the moving boundaries. We back-propagate gradients at the solid-fluid interface to extend gradient computation to the geometry iso-surface. Last, we provided an optimization framework to efficiently search the design space, considering the underlying fluid dynamics and boundary conditions.

Our pipeline features a low-dimensional parametric geometry representation capable of expressing complex shapes and a differentiable Navier-Stokes simulator with geometry gradient computation for predicting dynamic fluid behavior in response to control signals. In addition, our pipeline leverages gradient-based optimization for efficient design space exploration, co-optimization of the device geometry and control, and accurate performance evaluation of the design under dynamic flows.

The resulting pipeline supports gradient-based co-design of device geometry and control, as well as efficient evaluation of system performance under time-varying flow. To demonstrate the scope of the framework, this chapter considers a range of applications in robotics and engineering, including fluidic amplification and switching, flow modulation, system identification, and closed-loop control of systems such as water gates and artificial-heart-inspired devices. More broadly, this chapter extends the differentiability-aware perspective developed in the previous chapter from static flow design to dynamic fluidic systems, showing how simulation can support not only geometry optimization but also control and co-design in time-dependent settings.

4.2 Method

4.2.1 Pipeline Overview

We present an overview of our method in Fig. 4.1. Our pipeline defines the designs with a low-dimensional parametric geometry representation (Sec. 4.2.2). The behavior and performance of the design in the fluid environment are evaluated by a dynamic differentiable Navier-Stokes simulator (Sec. 4.2.3). Both components are embedded in a gradient-based optimization framework that co-optimizes both the geometric design and the control signal until convergence.

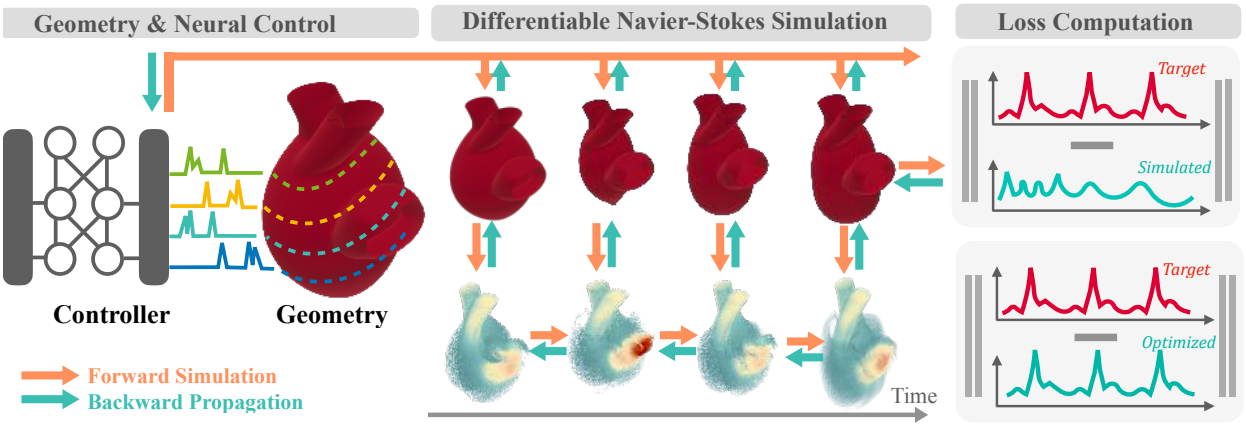
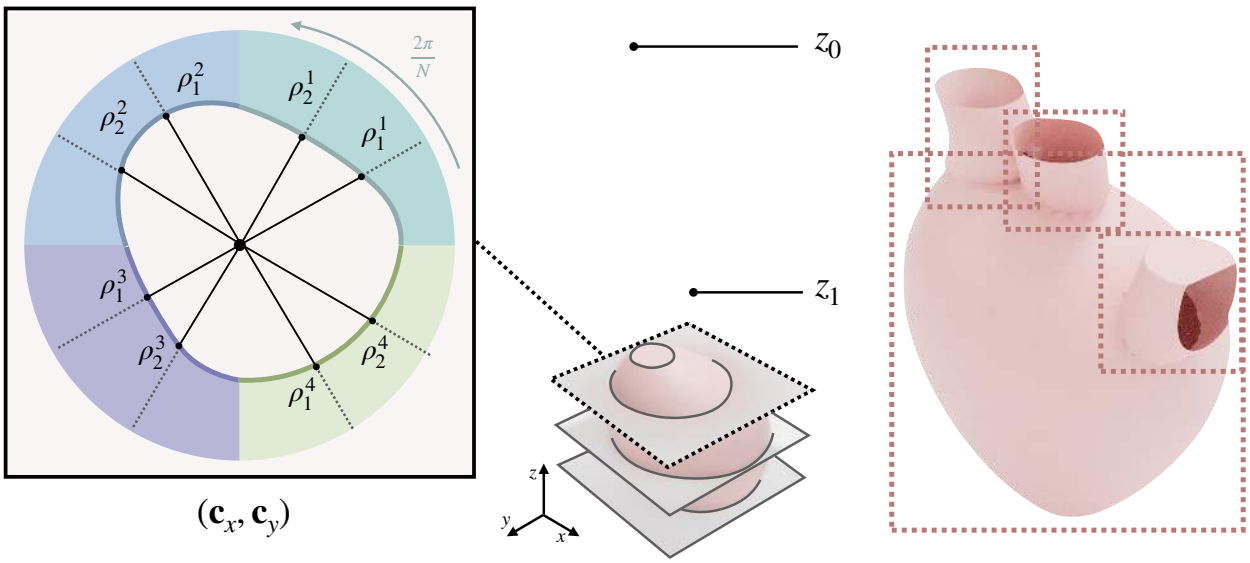


Figure 4.1: Pipeline Overview. (1) Our pipeline starts with an initial parametric geometry and a neural network parameterized controller. (2) The fluid dynamics is then simulated using a dynamic Navier-Stokes solver. (3) The performance of the design and control is evaluated using a loss function, the gradients of which are then back-propagated through our end-to-end differentiable framework. (4) The gradient-based optimization iteratively improves the geometry and control to achieve the task goal. This pipeline allows for efficient geometry and control co-optimization.

4.2.2 Geometry Representation

We represent our geometry with a low-dimension representation. Take the illustration in the inset figure as an example, here we introduce the representation on a high level. We parameterize a closed 2D surface using its center $\mathbf{c}$ and a set of connected Bezier curves with their control points define in polar coordinates ρ_i for $i \in [1, 2, \ldots, 2N]$, where every two control points define a 2D Bezier curve spanning $\frac{2\pi}{N}$ radians in the polar coordinate system. This representation offers a compact way of defining diverse geometries. We further parameterize a closed 3D surface using a list of 2D surfaces defining the key cross-sections of the geometry along an extrusion axis z of the local object frame, where each 2D surface is parameterized as described above. The parameterization includes $z = z_0$ and z_1, which determines the Z plane of the first and last cross-section, along with the parameters for each key 2D cross-section, which are assumed to be evenly spaced between $z \in [z_0, z_1]$.The continuous geometry interpolates the key cross-sections along the z-axis. Finally, we construct more complex 3D geometries using operations from Constructive Solid Geometry (CSG): Union and intersection, which allows us to define a 3D parametric heart model using the union of four sub-geometries.

2D Geometry We define a closed 2D surface using N connected cubic Bezier curves parameterized in the polar coordinate frame. A point $\mathbf{c}$ is defined on the surface to establish the center of a polar coordinate frame. Each Bezier curve spans an arc of $\frac{2\pi}{N}$ radians on the polar coordinate plane, and its control points are symmetrically placed, each at an angular displacement of $\frac{2\pi}{3N}$ radians from their corresponding curve endpoint. The shape of each Bezier curve, and consequently the overall surface, is manipulated via two scalar



a) Closed 2D Surface **b) Closed 3D Surface** **c) 3D Geometry**

parameters, ρ_1 and ρ_2, dictating the polar coordinate distance of the two control points. The i-th cubic Bezier curve is defined by two control points $\mathbf{p}_0 = (\rho_1^i \cos\theta, \rho_1^i sin\theta) + \mathbf{c}$ and $\mathbf{p}_1 = (\rho_2^i \cos\theta, \rho_2^i sin\theta) + \mathbf{c}$, where $\mathbf{c}$ is the reference center point. The endpoints $\mathbf{e}_0$, $\mathbf{e}_1$ are computed by ensuring $\mathbf{e}_1^i = \mathbf{e}_0^{(i+1)\%N}$ and colinearity of each pair of $\mathbf{p}_1^i, \mathbf{e}_1^i, \mathbf{p}_0^{(i+1)\%N}$. This representation offers a compact way of defining diverse geometries.

3D Geometry We parameterize a closed 3D surface using 2D surfaces defining the key cross-sections of the geometry along the z-axis of the local object frame, where each 2D surface is parameterized by N cubic Bezier curves. The parametrization includes $z = z_0$ and z_1, which determines the Z plane of the first and last cross-section and the parameters for each key cross-section. The $i-th$ key cross-section is defined by the center $\mathbf{c}^i$ and control point parameters ρ_1^i, ρ_2^i for each of $i \in [1, 2, \ldots, N]$. The key cross-sections are assumed to be evenly spaced along the z-axis. Then, given $z_0 \leq z \leq z_1$, the cross-section of the closed surface at $Z = z$ is defined by interpolating the centers and control points of all key cross-sections using the interpolation scheme

$$\begin{cases} \mathbf{c}_z = \sum_{i=1}^{n} \boldsymbol{c}^i (\frac{z - z_0}{z_1 - z_0})^i & \text{(4.1a)} \\ \rho_j = \sum_{i=1}^{n} \boldsymbol{\rho}_j^i (\frac{z - z_0}{z_1 - z_0})^i & \text{(4.1b)} \end{cases}$$

4.2.3 Differentiable Navier-Stokes Simulation

Our fluid dynamics is governed by the incompressible Navier-Stokes equations. These consist of the momentum equation (Eq. 4.2a), accounting for temporal changes in velocity ($\boldsymbol{u}$), advective acceleration, viscous dissipation, and pressure (p) gradient forces for an incompressible fluid

with fluid density ρ and kinematic viscosity ν. The incompressibility condition (Eq. 4.2b) requires the divergence of the velocity field must be zero to enforce the conservation of mass:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + \nu\nabla^2\boldsymbol{u} - \dfrac{1}{\rho}\nabla p, & \text{(4.2a)} \\ \nabla\cdot\boldsymbol{u} = 0 & \text{(4.2b)} \end{cases}$$

Numerical Simulation

We build the fluid simulator by leveraging the operator-splitting method [90][91]. A single simulation step comprises three sub-steps: advection, viscosity, and projection. The simulation domain is discretized on a standard Marker-and-Cell (MAC) grid [92], with pressures stored at cell centers and velocities at cell faces. By employing the finite-difference scheme on the MAC grid cells and faces, we construct the matrix $\frac{1}{\Delta x}\mathbf{G}$ for gradient operator and its negative transpose $-\frac{1}{\Delta x}\mathbf{G}^T$ for divergence operator. In the following sections, capital letters will refer to matrices or the flattened vectors induced by the fields denoted by the corresponding lowercase letters.

Advection We employ the semi-Lagrangian advection scheme, where the advected velocity field $\tilde{\mathbf{U}}^{n+1}$ is a linear interpolation of the velocity field $\mathbf{U}^n$. The interpolation position function is a function of $\mathbf{U}^n$ can be put into a matrix form $\mathbf{B}$, which results in:

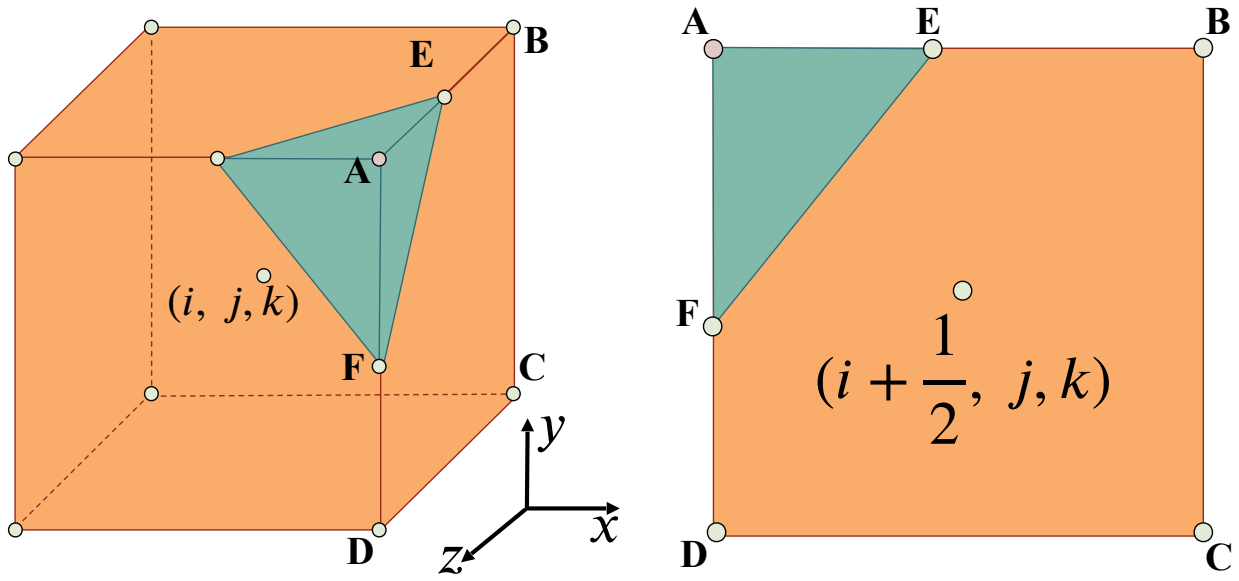
$$\tilde{\mathbf{U}}^{n+1} = \mathbf{B}(\mathbf{U}^n)\mathbf{U}^n. \tag{4.3}$$

Viscosity For incompressible fluid with a constant viscosity coefficient, the viscous force density is the product of the Laplacian of velocity and the viscosity coefficient. For each axis, the Laplacian of the corresponding velocity component is calculated on grid points using the finite difference method:

$$\hat{\mathbf{U}}^{n+1} = \left(\mathbf{I} - \frac{\nu\Delta t}{\Delta x^2}\mathbf{G}^T\mathbf{G}\right)\tilde{\mathbf{U}}^{n+1}. \tag{4.4}$$

Projection The projection step ensures the incompressibility of the fluid. Solid unaligned with the grid may intersect with grid faces, which can be captured with a cut-cell method [93]. We introduce α^{n+1} to represent the fluid proportion of a grid face. The solid's signed distance function (SDF) ϕ^{n+1} and the velocity $\boldsymbol{u}_s^{n+1}$ can derived from the solid geometry. We first use marching cube to compute the geometry zero contour. Next, we identify the intersection points between this contour and the grid faces and compute α^{n+1} based on ϕ. For instance, in the inset figure, grid face $(i+\frac{1}{2}, j, k)$ is cut by the contour, then $\alpha^{n+1}_{i+\frac{1}{2},j,k} = \frac{S_{AEF}}{S_{ABCD}} = \frac{1}{2}\cdot\frac{|AE|}{|AB|}\cdot\frac{|AF|}{|AD|} = \frac{1}{2}\cdot\frac{\phi_A^{n+1}}{\phi_A^{n+1}-\phi_B^{n+1}}\cdot\frac{\phi_A^{n+1}}{\phi_A^{n+1}-\phi_D^{n+1}}$.

The volume change rates for fluid and solid at grid cell (i, j, k), denoted as $\gamma^{n+1}_{f,i,j,k}$ and $\gamma^{n+1}_{s,i,j,k}$ respectively, equal to the sum of flux on the cell's surrounding faces, which can be calculated using α^{n+1}, fluid velocity $\boldsymbol{u}^{n+1}$, and solid velocity $\boldsymbol{u}_s^{n+1}$.



The incompressibility condition gives requires the sum of $\gamma_{f,i,j,k}^{n+1}$ and $\gamma_{s,i,j,k}^{n+1}$ to be zero, which gives

$$\frac{\Delta t}{\rho \Delta x}\mathbf{G}^T\mathbf{S}^{n+1}\mathbf{G}\mathbf{P}^{n+1} = \mathbf{G}^T\mathbf{S}^{n+1}\hat{\mathbf{U}}^{n+1} + \mathbf{G}^T(\mathbf{I} - \mathbf{S}^{n+1})\mathbf{U}_s^{n+1}. \tag{4.5}$$

where $\mathbf{S}^{n+1}$ is a diagonal matrix induced by α^{n+1}, and $\mathbf{P}$ is the pressure. After solving the linear system, the fluid velocity is updated based on the pressure values:

$$\mathbf{U}^{n+1} = \hat{\mathbf{U}}^{n+1} - \frac{\Delta t}{\rho \Delta x}\mathbf{G}\mathbf{P}^{n+1}, \tag{4.6}$$

Temporal Discretization of the Governing Equation

We build the fluid simulator by leveraging the operator-splitting method [90][91]. Each simulation step comprises of advection, viscosity, and projection.

Advection We employ the semi-Lagrangian advection scheme specified in (Eqs. 4.7 and 4.8) to propagate velocity through the fluid domain:

$$\frac{\tilde{\boldsymbol{u}}^{n+1/2} - \boldsymbol{u}^n}{\Delta t/2} = -\boldsymbol{u}^n \cdot \nabla \boldsymbol{u}^n, \tag{4.7}$$

$$\frac{\tilde{\boldsymbol{u}}^{n+1} - \boldsymbol{u}^n}{\Delta t} = -\tilde{\boldsymbol{u}}^{n+1/2} \cdot \nabla \boldsymbol{u}^n. \tag{4.8}$$

Viscosity For incompressible fluid with a constant viscosity coefficient, the viscous force density is equivalent to the product of the Laplacian of velocity and the viscosity coefficient. We employ explicit time integration to update the fluid velocity in response to the viscous force.

$$\frac{\hat{\boldsymbol{u}}^{n+1} - \tilde{\boldsymbol{u}}^{n+1}}{\Delta t} = \nu \nabla^2 \tilde{\boldsymbol{u}}^{n+1} \tag{4.9}$$

Projection The projection step involves an update of the pressure and the velocity field (Eq. 4.10a) to ensure the satisfaction of the incompressibility condition (Eq. 4.10b).

$$\begin{cases} \dfrac{\boldsymbol{u}^{n+1} - \hat{\boldsymbol{u}}^{n+1}}{\Delta t} = -\dfrac{1}{\rho}\nabla p^{n+1}, & \text{(4.10a)} \\ \nabla \cdot \boldsymbol{u}^{n+1} = 0. & \text{(4.10b)} \end{cases}$$

On boundaries, the pressure is regulated by two conditions: the Dirichlet boundary condition (Eq. 4.11a) and the non-penetrating Neumann boundary condition (Eq. 4.11b) given computed geometry velocity $\boldsymbol{u}_s^{n+1}$:

$$\begin{cases} p^{n+1} = 0, \quad \boldsymbol{x} \in \partial\Omega_f^{n+1}, & \text{(4.11a)} \\ \boldsymbol{u^{n+1}} \cdot \boldsymbol{n} = \boldsymbol{u_s^{n+1}} \cdot \boldsymbol{n}, \quad \boldsymbol{x} \in \partial\Omega_b^{n+1}. & \text{(4.11b)} \end{cases}$$

The pressure field for the subsequent time step p^{n+1} is determined by solving the resultant Poisson equation (Eq. 4.12).

$$\frac{\Delta t}{\rho}\nabla^2 p^{n+1} = \nabla \cdot \hat{\boldsymbol{u}}^{n+1}. \tag{4.12}$$

Back-propagation through Time

We construct our back-propagation algorithm to mirror the sequence of operations carried out in the forward pass but in a reversed order.

Given the gradients of the loss function J with respect to the velocity field $\boldsymbol{u}$ at time step $n+1$, denoted by $\frac{\partial J}{\partial \mathbf{U}^{n+1}}$, our goal is to compute the corresponding gradients at time step n, $\frac{\partial J}{\partial \mathbf{U}^{n}}$.

Projection We begin by reversing the projection step to back-propagate $\frac{\partial J}{\partial \mathbf{U}^{n+1}}$ to derive $\frac{\partial J}{\partial \mathbf{P}^{n+1}}$ and $\frac{\partial J}{\partial \hat{\mathbf{U}}^{n+1}}$. Back-propagating through Eq.4.6 gives

$$\frac{\partial J}{\partial \mathbf{P}^{n+1}} = -\frac{\Delta t}{\rho \Delta x}\frac{\partial J}{\partial \mathbf{U}^{n+1}}\mathbf{G}. \tag{4.13}$$

We can back-propagate the adjoint of Eq. 4.5 w.r.t $\hat{\mathbf{U}}^{n+1}$ by defining the adjoint variable $\mathbf{y}$ and derive

$$\frac{\partial J}{\partial \hat{\mathbf{U}}^{n+1}} = \frac{\partial J}{\partial \mathbf{U}^{n+1}} + \mathbf{y}\mathbf{G}^T\mathbf{S}^{n+1}, \tag{4.14}$$

where $\mathbf{A} = \frac{\Delta t}{\rho \Delta x}\mathbf{G}^T\mathbf{S}^{n+1}\mathbf{G}$, $\mathbf{b} = \mathbf{G}^T\mathbf{S}^{n+1}\hat{\mathbf{U}}^{n+1} + \mathbf{G}^T(\mathbf{I} - \mathbf{S}^{n+1})\mathbf{U}_s^{n+1}$, and $\mathbf{y}$ is computed by solving the linear system $\mathbf{A}\boldsymbol{y}^T = (\frac{\partial J}{\partial \mathbf{P}^{n+1}})^T$.

Viscosity and Advection Back-propagating through viscosity and advection simply involves back-propagating $\frac{\partial J}{\partial \hat{\mathbf{U}}^{n+1}}$ through Eq. 4.4 and Eq. 4.3, which allows us to derive:

$$\frac{\partial J}{\partial \mathbf{U}^{n}} = \frac{\partial J}{\partial \hat{\mathbf{U}}^{n+1}}\left(\mathbf{I} - \frac{\nu \Delta t}{\Delta x^2}\mathbf{G}^T\mathbf{G}\right).(\frac{\partial \mathbf{B}}{\partial \mathbf{U}^n}\mathbf{U}^n + \mathbf{B}(\mathbf{U}^n)). \tag{4.15}$$

The above equations provide the outline of the back-propagation process through a single time step of from time step $n+1$ to n. To compute the gradients of the loss function J at any time step, we iterate the back-propagation process over the full sequence of time steps.

Back-propagation through Geometry

The parametric geometry affects simulation through the solid-fluid boundary during the projection step in Eq. 4.5. Specifically, the SDF of the geometry ϕ^{n+1} affects the volume matrix $\mathbf{S}^{n+1}$ and the velocity (in the case of moving geometry) of the geometry $\mathbf{U}_s^{n+1}$ affects the boundary condition. We can back-propagate $\frac{\partial J}{\partial \mathbf{P}^{n+1}}$ w.r.t these two parameters to derive

$$\begin{cases} \dfrac{\partial J}{\partial \mathbf{S}^{n+1}} = \boldsymbol{y}\mathbf{G}^T(\hat{\mathbf{U}}^{n+1} - \mathbf{U}_s^{n+1}) + \boldsymbol{y}\dfrac{\partial \mathbf{A}}{\partial \mathbf{S}^{n+1}}\mathbf{P}^{n+1}, & \text{(4.16a)} \\ \dfrac{\partial J}{\partial \mathbf{U}_s^{n+1}} = -\boldsymbol{y}\mathbf{G}^T\mathbf{S}^{n+1}. & \text{(4.16b)} \end{cases}$$

Further back-propagating $\frac{\partial J}{\partial \mathbf{S}^{n+1}}$ first through the SDF ϕ then through the distance computation and $\frac{\partial J}{\partial \mathbf{U}_s^{n+1}}$ through geometry velocity function allows us to optimize through the geometry iso-surface.

Neural Fluid Control

We can train neural-network parameterized closed-loop fluid controllers with gradients fully computed at both geometry and velocity throughout time. We parameterize our controllers with a two-layer MLP. The controller takes as input the observation of the fluid velocity field at each frame and outputs dynamic control signals that affect the geometry through our parametric geometry presentation, which further affects the flow field. Our fully differentiable framework allows gradient-based methods to train the controller efficiently. We implemented the backbone of our code in C++ and CUDA for computational efficiency. We derived gradients for the geometry and simulation module analytically, then exposed the differentiable simulation framework through pybind11 [94] to enable seamless integration with deep learning libraries, which in our case is PyTorch [95].

4.3 Benchmarks and Applications

In this section, we introduce our fluidic design benchmarks and environments. We build our environments using the standard protocol in the gym to facilitate learning practices. We present six fluidic design and control applications (Fig. 4.2) to assess the effectiveness of our computational pipeline for fluidic system design and learning. Initial conditions are set for all optimizations using randomly sampled values. We use Adam as our optimizer. We summarize the simulation configuration and optimization configuration as well as relevant statistics in Table 4.1.

Table 4.1: Application Specifications. We summarize the simulation and optimization configuration for the design applications shown in Sec. 4.3 and report the initial and optimized loss. We note that because our implementation adopts CFL condition for numerical stability during simulation, the actual steps simulated and back-propagated are higher than the numbers shown in "# Frames".

	Resolution	**# Frames**	**# Param.**	$\partial L_f/\partial$**Design**	$\partial L_f/\partial$**Control**	**Loss** (L_f)	
						Initial	**Optimized**
Amplifier	64×64	40	5	✓		13.401	0.005
Switch	64×64	120	10	✓	✓	13.162	1.893
Shape Identifier	128×128	10	10	✓		70.152	0.759
Flow Modulator	$40 \times 40 \times 40$	100	34	✓	✓	6.118	0.069
Neural Gate	$40 \times 40 \times 40$	50	4.5k		✓	7.318	0.000
Neural Heart	$48 \times 48 \times 48$	180	7.1k	✓	✓	1.086	0.004

Figure 4.2: Applications Overview. In each application, the blue dashed line represents the inlet, the red dashed line indicates the outlet, the white arrows show the flow direction, and the orange shapes and arrows denote the geometry and its motion direction.

4.3.1 Application Overview

Amplifier This design problem aims to amplify a parallel horizontal inflow by three times from an initial velocity of 5 units to 15 units. The device boundaries are parameterized as two symmetrically placed cubic Bezier curves. The design variables are the control points and endpoints of the two curves. The loss function is defined as the last frame L2 norm of the difference between the target and optimized fluid velocity norm. We visualize the initial and optimized designs in Fig. 4.3a. We overlay the design and the corresponding velocity field (colored by the norm) for both iterations.

Shape Identifier This application provides an example of system identification in a fluid environment by identifying the shape and position of a geometry, given observations of the flow field. We randomly initialize the geometry in the domain. We define the loss function as the sum of the L2 norm of the velocity field difference to the observed ground-truth flow field across time. The optimization successfully reconstructs the shape and its position with random initialization (See Exp. 4.4.2).

Switch This application simultaneously optimizes the geometry and the constant rotational speed of a 2D switch, allowing dynamic regulation of outlet flow velocity. A horizontal inflow at the left interacts with the switch, splitting into two distinct streams towards the right

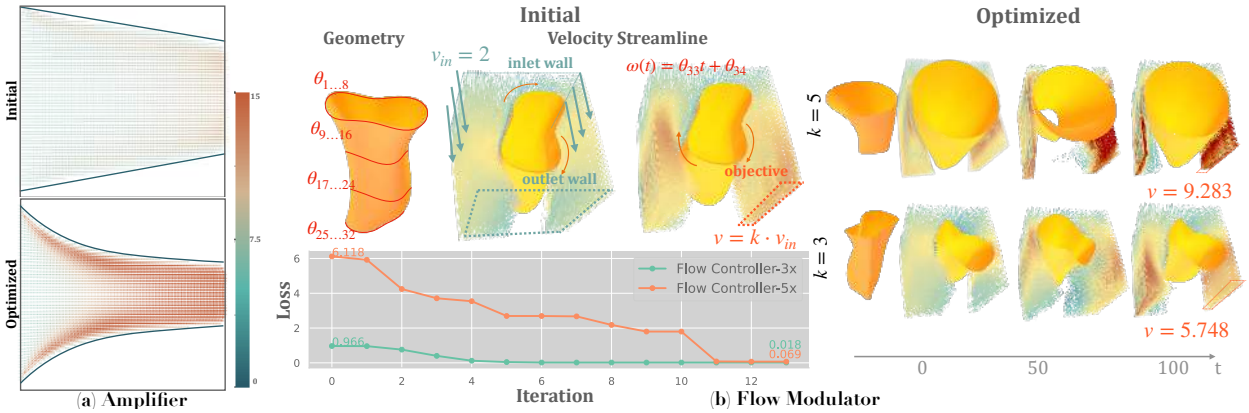


Figure 4.3: (a) Visualization of *Amplifier*. (b) Visualization of *Flow Modulator*.

side. The goal is to let the time-dependent average velocity norm of the upper stream align with a predetermined flow profile. We define the loss function as the sum of the L2 norm of the difference between the average velocity norm of the fluid and the target velocity at each frame. The optimized design and control successfully generate a linearly increasing upper stream velocity norm profile, matching the specified target. We visualize the initial and optimized design for the switch application in Fig. 4.4.

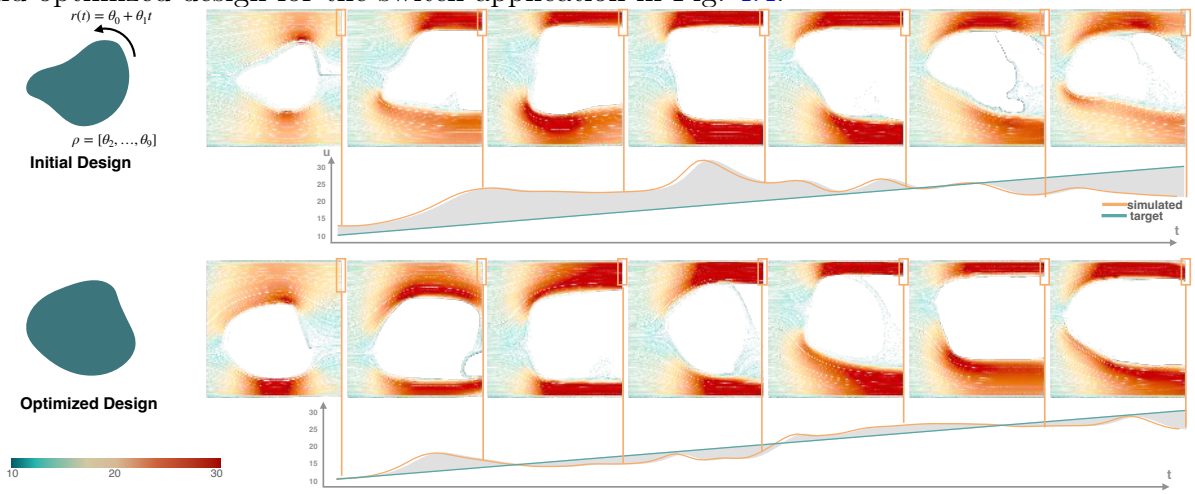


Figure 4.4: Fluidic Switch. The switch rotates dynamically across time (dotted lines). The shape of the switch is parameterized as a 2D Polar Bezier, whose parameters, along with the parameters of the rotation signal are subject to optimization. The top and bottom of the illustration visualize information from the initial and optimized iteration respectively. For each iteration, we visualize the design geometry (left) and corresponding streamlines of the flow field (right) at 7 key-frames evenly sampled across time. We additionally plot the target (green) and outlet velocity norm profile (orange) across time and visualize their difference in grey shaded area.

Flow Modulator This application optimizes the geometry and control of a rotating 3D flow controller to achieve a target average outlet flow ($k\times$ inflow) at the domain's right boundary by the end of the simulation. The controller's geometry is parameterized by

four 3D cross-sections, with rotation controlled by a sinusoidal function. We define the loss as the L2 norm of the difference between the average outlet and target velocity at the final frame. Fig. 4.3b illustrates the application: the top left shows initial parameters and application specification, bottom left shows optimization trajectories, and the right visualizes the optimized geometry and velocity streamline with two variants ($k = 3, 5$).

Neural Gate Controller We learn a closed-loop controller for a 3D fluid gate moving horizontally. The controller is parameterized as a two-layer MLP. It observes the current outflow velocity through the gate and outputs the next frame motion offset to control the outflow velocity to match a target.

4.3.2 Scaling to Complex Fluid Fields: Artificial Heart

We showcase the scalability of our method through an artificial heart design and control learning application (Fig. 4.5). Artificial heart development is difficult due to the complex blood flow movement within the heart. This case study provides a first step in studying the heart's control strategies. We train a closed-loop controller that outputs the per-time-step contraction signal of the four muscles of a simplified heart model so that the outlet velocity matches a pre-defined target profile. The controller's states include temporal encoding of the current time step and the current outflow norm, and they are parameterized using a two-layer MLP. The heart's geometry is parameterized as the union of the two inlets, one outlet, and the heart chamber. In two variants of the application, one target flow profile is parameterized using a cosine curve (Fig 4.5 top), and one target flow profile mimics the shape of an electrocardiogram (Fig 4.5 bottom). We define the loss function as the sum of the L2 norm of the difference between the average velocity norm of the fluid and the target velocity at each frame. In both variants, the trained controller successfully outputs signals that generate blood flow that matches the target, demonstrating the effectiveness of our gradient-based optimization framework. We further visualize the rollouts of the trained controllers at Fig 4.5 left.

Figure 4.5: Artificial Heart. *Left*: visualization of the domain and the location of the muscles. *Middle*: Optimized control policy rollout visualization. *Right:* Optimization results visualization. The top and bottom diagrams visualize the cosine and the ECG target variants.

4.4 Experiments

4.4.1 Effects of Initialization on Optimization

This experiment studies the effect of random initialization in our fluid optimization tasks. Specifically, we conduct an experiment on the 3D heart controller task, which utilizes a neural network with 7,100 parameters. For this task, we initialized the network parameters with five different random seeds, tracking convergence under each condition. In the accompanying figure, we plot an extended version of the training trajectory, scaled logarithmically for better visualization, to compare the convergence of different random initializations. However, in practice, our method achieves the objective driven by the loss within tens of iterations, as is evident from the steep initial descent in the optimization curve. As shown in Fig. 4.6 left, the optimization consistently converges across all seeds, despite variations in initial network behavior. This consistency indicates the robustness of our method to random initialization even in high-dimensional optimization spaces, supporting its application to complex tasks in differentiable physics. These curves offer valuable guidelines for practitioners using our method in their deployments: the gradients from our approach are robust to hyper-parameters and scalable to high-dimensional optimization problems.

4.4.2 Gradient-Based vs Gradient-Free Optimization

We study the effectiveness of our gradient-based method against gradient-free optimization methods in the *Neural Heart* task (Fig. 4.6 right). We choose Proximal Policy Optimization (PPO) [96] as the baseline for reinforcement learning and CMA-ES [97] for evolution strategies. This environment is particular challenging due to the sensitivity of the bloodflow to the control signal changes across time, which could result in large flow field change if adjacent rollouts have large changes. We initialize all methods to output stochastic control signals of small noise for stable initial simulation. Our gradient-based method quickly converges to near zero after 60 epochs, while both gradient-free methods struggle in this environment. We argue that the rapid and successful convergence stems from the clear gradient provided by our method. Note that our differentiable optimization pipeline depends on the differentiability of the loss function (e.g., the imitation loss we used). This can be problematic if the objective is too complex to be characterized in a differentiable manner, in which case gradient-free methods are better alternatives. However, our framework will still excel due to its outstanding forward simulation speed, which we will elaborate next.

4.4.3 Time Performance Profiling and Comparison with PhiFlow

In this experiment, we demonstrate the performance efficiency of our framework through a comparison with PhiFlow. While PhiFlow operates with a TensorFlow-GPU backend, our framework is implemented in CUDA C++ and features a high-performance Geometric-Multigrid-Preconditioned-Conjugate-Gradient (MGPCG) Poisson solver [98]. To address the needs of differential operators and interpolations, which require access to neighboring cells in all directions, we divide the simulation domain into cubic blocks, each corresponding to a CUDA block. When launching a CUDA kernel, simulation data for each block is first

Table 4.2: Time Performance. Our method achieves one order of magnitude speedup across all resolutions compared to PhiFlow in both forward simulation and backward gradient propagation.

	Forward			Backward		
Resolution	**PhiFlow (s)**	**Ours (s)**	**Speedup**	**PhiFlow (s)**	**Ours (s)**	**Speedup**
32×32×32	1.282	**0.024**	**53.4**×	1.546	**0.095**	**16.3**×
40×40×40	1.741	**0.039**	**44.6**×	1.983	**0.121**	**16.4**×
48×48×48	2.227	**0.068**	**32.8**×	2.412	**0.158**	**15.3**×
64×64×64	3.145	**0.105**	**30.0**×	4.094	**0.301**	**13.6**×

Table 4.3: Memory and time performance comparison with DiffTaichi

	Memory (MB)		Forward Time (s)		Backward Time (s)	
Resolution	**DiffTaichi**	**Ours**	**DiffTaichi**	**Ours**	**DiffTaichi**	**Ours**
32 × 32 × 32	685	**292**	0.081	**0.024**	0.074	**0.027**
40 × 40 × 40	1136	**308**	0.146	**0.039**	0.133	**0.041**
48 × 48 × 48	1805	**322**	0.228	**0.068**	0.183	**0.064**
64 × 64 × 64	5005	**405**	0.435	**0.105**	0.363	**0.117**

loaded into shared memory, allowing efficient computation directly in shared memory and reducing global memory accesses. Additionally, to increase memory throughput, each block's data is stored consecutively in global memory. Our matrix-free MGPCG solver has a faster convergence rate than PhiFlow's Conjugate Gradient solver and uses a hierarchical grid data structure, with custom CUDA kernels for prolongation and restriction operations between coarse and fine grids.

We benchmark both the one-step forward simulation time and gradient back-propagation time at different resolutions, as shown in Table 4.2. The experiment runs on a workstation with an NVIDIA RTX A6000 GPU, where our framework consistently outperforms PhiFlow by an order of magnitude across all resolutions, benefiting both gradient-based and gradient-free optimization techniques. This performance improvement is particularly advantageous in fluid simulation applications, including robotics and video generation. Additionally, our system's gym protocol compatibility [87] makes it straightforward for practitioners to integrate and test our library.

4.4.4 Memory and Time Performance Profiling and Comparison with DiffTaichi

Here we compare our solver with DiffTaichi, a differentiable programming framework, to highlight the benefits of our approach in terms of scalability and efficiency. Our method, designed specifically for differentiable fluid simulation, uses manually derived gradients, avoiding the need to store intermediate computational graph at each timestep, unlike DiffTaichi, which

Figure 4.6: Ablation Studies. *Left:* Optimization trajectories for *Neural Heart* with 7100 parameters under different initialization. Iterations are visualized on a log scale. *Right:* Log scaled loss-iteration curves of our gradient-based method and other gradient-free optimization methods.

relies on automatic differentiation. Additionally, our adjoint derivation for the projection solve step is independent of solver iterations, making our approach well-suited for advection-projection fluid simulations. To demonstrate this, we implemented a Conjugate Gradient (CG) solver for the projection step in DiffTaichi and compared time and memory performance across four grid resolutions in a 3D optimization scenario (Table 4.3). Our results show that our solver requires substantially less memory, with up to 12 times less memory usage than DiffTaichi at $64 \times 64 \times 64$ resolution. This reduction in memory stems from eliminating the need to store intermediate values during each CG iteration, making our solver particularly suitable for high-resolution, long-term optimizations.

4.4.5 Fluid Solver Validation – Kármán Vortex Street

To further validate the performance of our solver, we conducted an additional experiment simulating the formation of a classic **Kármán Vortex Street**. This experiment was executed at a resolution of 512×1024 and illustrates the capability of our solver in capturing complex fluid dynamics phenomena. As shown in Fig. 4.7, we simulate a horizontal flow passing around a cylindrical obstacle at three distinct kinematic viscosity values: inviscid ($\nu = 0$), moderate viscosity ($\nu = 0.002$), and high viscosity ($\nu = 0.02$).

Each viscosity setting demonstrates the characteristic vortex shedding pattern associated with Kármán Vortex Street formation. These results confirm our solver's ability to replicate this well-known phenomenon and offer insights into the effect of varying viscosity on vortex behavior. This validation experiment supports the accuracy and versatility of the solver across different fluid conditions.

4.4.6 Gradient Stability and Solver Steps Statistics

Gradient stability in differentiable physics is a well-known challenge, particularly given the potential for gradient explosion or vanishing when gradients are accumulated across numerous solver steps. In our approach, however, we have not encountered significant issues with

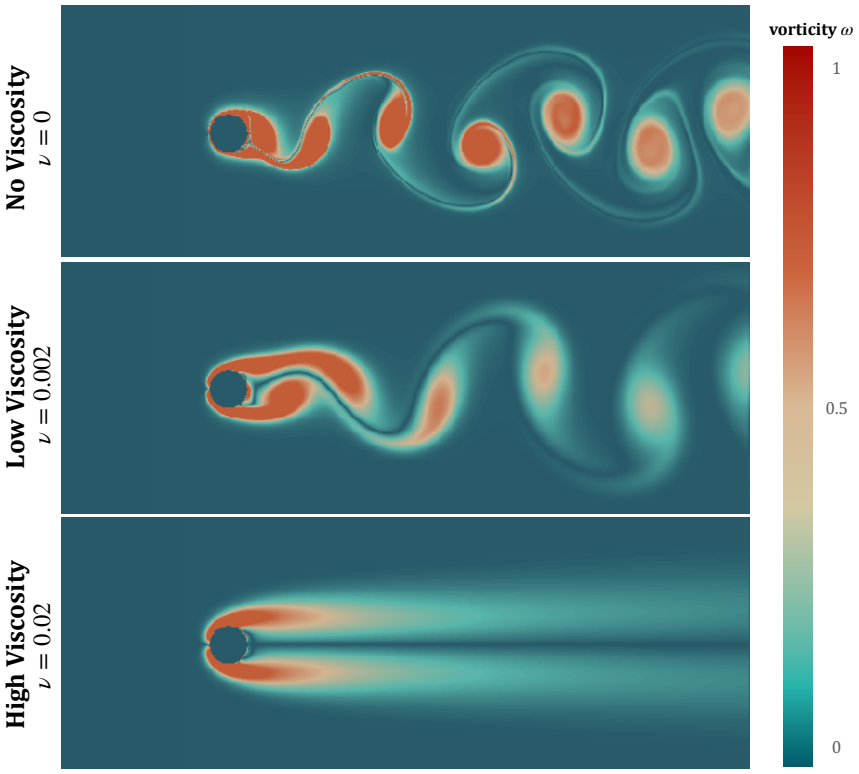


Figure 4.7: Solver Validation. Visualization of Karman Vortex Street under different viscosity conditions. Here, we illustrate the results of the classic Karman Vortex Street test for three different kinematic viscosity values (From top to down $\nu = 0.0$, $\nu = 0.002$ and $\nu = 0.02$), simulated using our differentiable simulator within a domain size 512×1024. Each figure visualizes the vortex patterns, and the results demonstrate how increased viscosity leads to a notable change in vortex formation and dissipation.

gradient stability. This stability is likely due to the accuracy of the gradients produced by our framework and the robustness of our numerical solver. To illustrate this, we provide gradient norm statistics over the full course of optimization for three tasks of varying complexity in Table 4.4, demonstrating consistent gradient magnitudes without evidence of explosion or vanishing. For additional robustness, our implementation includes gradient clipping with a threshold of 1.0, which can mitigate gradient explosion in particularly challenging scenarios. This technique ensures gradients remain within manageable limits and contributes to the overall stability of our optimization pipeline.

Furthermore, the actual number of solver steps required to advance between frames in our solver depends on the Courant-Friedrichs-Lewy (CFL) condition, which is maintained to ensure numerical stability. Additional statistics on solver steps over one optimization cycle across various tasks are provided in the upper portion of Table 4.4, offering further insight into the computational demands and stability characteristics of our approach.

Metric	Statistic	Shape Identifier	Heart 3D	Gate 3D
Steps	Mean, Std	(11, 0)	(54, 0)	(58, 0)
Gradient Norm	Min, Max	(0.46, 252.37)	(1.61, 820.82)	(5.13, 169.21)
	Mean, Std	(28.04, 38.00)	(65.96, 109.40)	(40.71, 34.36)

Table 4.4: Statistics of the gradient norm and step count over the full course of optimization for three tasks of varying complexity.

Table 4.5: Gradient Validation for Shape Identifier Task

	Gradient Values by Parameter (P1 to P6)					
	P1	**P2**	**P3**	**P4**	**P5**	**P6**
Analytic	0.048	0.394	0.314	0.046	0.133	0.067
Finite Diff	0.048	0.396	0.314	0.046	0.134	0.066
Abs Diff	-1.7e-4	-2.1e-3	5.2e-4	-3.4e-4	-1.4e-3	7.2e-4
Elem Err	0.004	0.005	0.002	0.007	0.011	0.011
	Gradient Values by Parameter (P7 to P11)					
	P7	**P8**	**P9**	**P10**	**P11**	
Analytic	0.087	0.008	-0.022	0.540	-0.058	
Finite Diff	0.086	0.008	-0.025	0.540	-0.059	
Abs Diff	2.1e-4	-9.6e-5	2.2e-3	2.4e-4	9.5e-4	
Elem Err	0.003	0.012	0.098	0.000	0.016	

4.4.7 Gradient Validation

To ensure the correctness of the gradients in our differentiable simulation framework, we validated the analytical gradients of all kernels, functions, and the entire simulation and optimization pipeline using finite difference approximations. Specifically, we employed the central difference method with a step size of 1.2×10^{-5} to approximate the gradients numerically and compared them with the analytical gradients calculated by our solver.

In this validation experiment, we consider the end-to-end gradients for the Shape Identifier Task, which involves optimizing over 11 parameters. The analytical gradients, finite difference gradients, their absolute differences, and element-wise errors are reported in Table 4.5. For this task, we observed a relative error of 0.0047 for the gradient vector, confirming the high accuracy of our analytical gradients.

4.5 Discussion and Conclusion

This chapter developed a fully differentiable pipeline for control and design of dynamic fluidic systems, addressing the challenges of geometry representation, differentiable fluid simulation, and co-design optimization. The framework combines a low-dimensional parametric geometry

representation with a differentiable Navier–Stokes simulator to predict fluid behavior under control inputs. Its effectiveness was demonstrated across a range of control and design tasks, from fluidic functional devices to artificial-heart-inspired control problems.

There are, however, several limitations and directions for future work. First, the current framework assumes the standard Navier–Stokes model and is therefore limited to Newtonian flow. Extending the method to non-Newtonian fluids or more general multiphysics interactions would broaden its applicability. Second, the pipeline relies on a parametric geometry representation, which may become restrictive in design spaces with high-dimensional or discontinuous changes, particularly when control design is coupled with topology optimization. Exploring alternative optimization strategies or surrogate models may therefore improve both efficiency and robustness. Finally, although the framework performs well across several simulated control and design tasks, additional validation against real-world physical experiments will be important for establishing its reliability and generalizability.

More broadly, this chapter extends the thesis from static fluidic design to dynamic fluidic systems, showing how differentiable simulation can support not only geometry optimization but also control and co-design in time-dependent settings.

Chapter 5

Neural Modular Physics and Data Adaptation

5.1 Motivation

Learning-based simulators have emerged as a powerful approach to modeling complex physical systems, where neural networks are usually end-to-end optimized from data as a whole to approximate object dynamics. Representative neural simulators include neural operators [21, 68–70] and Graph Neural Network (GNN) simulators [19, 64–67], which promise fast, flexible and differentiable simulations, showing promising results in fluid dynamics [61], material science [99], and beyond. However, the above neural simulators typically suffer from key limitations regarding physical soundness. Specifically, neural simulators often behave as an indivisible function, where the only accessible physics quantity is the model output, thereby leaving no access to intermediate physical quantities (e.g., internal forces, stress) essential for simulation. Such opacity seriously damages simulator interpretability and also makes it difficult to enforce known physical constraints. Although some physics-informed neural networks [23, 24, 100–103] can explicitly optimize the model with physics equations, they usually suffer from serious training difficulties [18, 25] and are mainly limited to simple equations and scenarios [17, 26], which are hard to support the simulation of complex long-horizon dynamics focused on in this chapter.

A physical trajectory goes far beyond a sequence of states following a vague data distribution. Taking elastic body dynamics as an example, the dynamics at each step is governed by global conservation laws, local geometric constraints, and material-specific constitutive relations, which together confine motion to a narrow subset of mathematically possible state space, necessitating the physical soundness of simulators. Unlike recent neural simulators, all the above-mentioned physical laws are carefully maintained in traditional numerical simulators, such as finite element methods [10]. In general, these classical simulators work modularly, which decomposes complex physical dynamics into several sub-modules and defines the interaction among intermediate quantities based on physical prior knowledge. The modular computation flow in classical simulators not only decomposes the complex dynamics into easier sub-processes but also offers a flexible interface to constrain physical quantities for strict physical soundness. This observation motivates introducing *modularity* of classical

simulators into learning-based simulators.

Previous researchers have explored some hybrid neural-physics simulators by substituting one of the modules in classical simulators with learnable neural networks, such as replacing discretization stencils [15] or constitutive laws [22], which can be viewed as incidental and initial explorations of the modular motivation. However, since pure numerical methods are less adaptable in scenarios with unknown information (e.g., environment frictional forces), these hybrid approaches only achieve a compromise performance, which sacrifices scenario flexibility of data-driven neural simulators for better physical correctness. These piecemeal and unsatisfactory approaches naturally raise a question: *could we design neural simulators with a thorough modular architecture, maintaining both data-driven flexibility and physical soundness?* After a comprehensive empirical investigation, we find that the correct answer for modularization of physics simulation is far beyond simple substitution, which requires elaborate design to faithfully ensure physical alignment with numerical simulators and avoid *collapse* in complex modular architecture [73]. Here, "collapse" refers to confused module capability, a foundation issue of modular neural networks [74], and will cause undesirable module behavior and overall performance.

In this chapter, we propose a *Neural Modular Physics* (NMP) framework that embraces a modular learning philosophy for neural elastic simulation, which involves a specialized architecture and a modular physics training strategy. Specifically, our approach decomposes physical simulation into well-defined modules, replacing key components such as constitutive models and time integration schemes with specialized neural networks. This physically aligned modular design exposes essential physical quantities in the middle of the neural simulator, thereby enabling direct supervision of intermediate quantities and flexible interchange between neural and numerical implementations. Additionally, to avoid the potential collapse of the modular architecture, NMP employs a two-stage training strategy, where components are first trained independently, supervised by intermediate physics quantities of numerical simulators, before being fine-tuned together, enabling the *specialization* and stricter physical alignment of each module. Furthermore, benefiting from modular design, NMP achieves favorable flexibility in enforcing physical constraints. By utilizing exposed intermediate physical quantities, such as the deformation gradient of an elastic body, NMP enables more detailed physics constraints, such as local volume preservation, which cannot be accomplished in previous end-to-end neural simulators, demonstrating the potential of this new modular learning paradigm. We demonstrate these advantages through comprehensive experiments across diverse elastic simulation tasks. To summarize our contributions:

- We propose a new thorough modular neural framework *Neural Modular Physics* for elastic simulation. Every classical sub-system can be replaced by a neural module with well-defined physical interfaces, enabling direct supervision of intermediate quantities, interchangeability with physics-based modules and flexible physical constraints.

- A specialized architecture and modular physics training strategy are presented to ensure better physical alignment with numerical methods and avoid collapse in complex modular architectures, which successfully guarantees the stable training and module specialization.

- NMP demonstrates strong generalization to unseen initial conditions and resolutions,

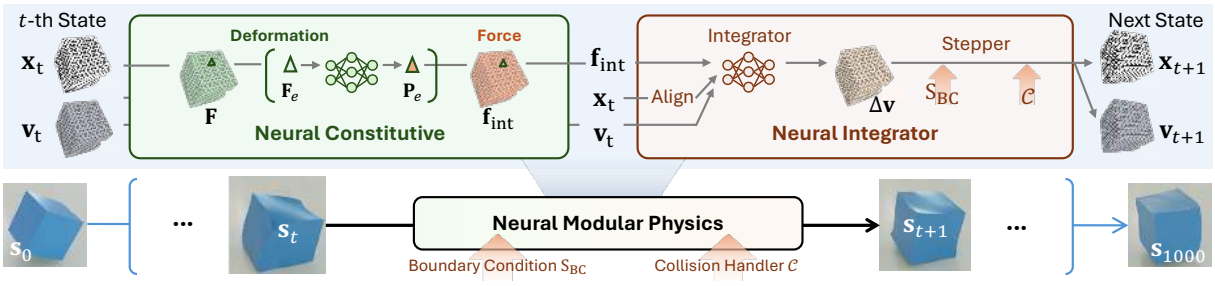


Figure 5.1: NMP modularizes elastic physics simulation into two parts: neural constitutive module to compute internal forces $\mathbf{f}_{\text{int}}$, and neural integration module to evolve system state $\mathbf{s}_t = \{\mathbf{x}_t, \mathbf{v}_t\}$ and handle traditional boundary conditions $\mathbf{S}_{BC}$ and collision $\mathcal{C}$.

delivering accurate long-horizon rollouts that outperform advanced neural simulators and showing better flexibility in scenarios with unknown information than hybrid simulators.

5.2 Neural Modular Physics

This chapter proposes a modular framework for elastic simulation to improve scenario flexibility and physical soundness. We introduce Neural Modular Physics (NMP), which enforces physical alignment and module specialization through a dedicated modular architecture and training strategy. We first review insights from classical simulators, then describe the formulation and training of NMP.

5.2.1 Inspirations from Classical Simulators

Since module specialization is key to neural modular networks [73], identifying appropriate model modules is essential. Unlike language or vision tasks, physical simulation benefits from long-standing references in classical numerical simulators. We therefore follow the computation process of classical simulators, which naturally defines heterogeneous modules and provides physically meaningful interfaces for subsequent optimization.

Specifically, the motion of a deformable object obeys Newton's second law $\mathbf{M}\ddot{\mathbf{x}} = \mathbf{f}_{\text{int}}(\mathbf{x}) + \mathbf{f}_{\text{ext}}$, where $\mathbf{x} \in \mathbb{R}^{3n}$ stacks the nodal positions of a mesh with n vertices, $\mathbf{M}$ is the (lumped) mass matrix, $\mathbf{f}_{\text{int}}$ denotes internal elastic forces, and $\mathbf{f}_{\text{ext}}$ collects external loads such as gravity or contact. A standard numerical method based on discretization under tetrahedral or hexahedral elements involves the following computation: (i) first computes the deformation gradient $\mathbf{F}$ inside each element from nodal displacements, then assembles element forces into the global system, (ii) along with a time integration to simulate dynamics. The above computation obviously derives two decoupled modules: spatial force computation (known as the Constitutive law) and time integration. Inspired by this observation, NMP modularizes the elastic simulation into *neural constitutive module* and *neural integration module*, which will be detailed next.

5.2.2 Elastic Physics Modularization

Drawing inspiration from classical simulators, we factorize the traditional computation process into two successive neural modules, each aligned with a well-defined physical sub-process as presented in Fig. 5.1.

Specifically, given system state $\mathbf{s}_t$ with vertex position $\mathbf{x}_t$ and velocity $\mathbf{v}_t$, neural constitutive module maps deformation gradient $\mathbf{F}$ computed from nodal displacements to internal forces $\mathbf{f}_{\text{int}}$, while neural integration evolves $\mathbf{s}_t$ and completes each step with boundary condition enforcement ($\mathbf{S}_{BC}$) and collision handling ($\mathcal{C}$), maintaining traditional simulator's computational flow (Alg. 1).

Algorithm 1: Neural Modular Physics

Input: $\mathbf{s}_t = \{\mathbf{x}_t, \mathbf{v}_t\}$, $\mathbf{S}_{BC}$, $\mathcal{C}$
Output: $\mathbf{s}_{t+1} = \{\mathbf{x}_{t+1}, \mathbf{v}_{t+1}\}$
// Neural constitutive module
foreach *element* e **do**
  Compute deformation gradient $\mathbf{F}_e$ from $\mathbf{x}_t$
  Predict stress $\mathbf{P}_e = f_\theta(\mathbf{F}_e)$
  Compute element forces $\mathbf{f}_{\text{int}}^e = \mathbf{B}_e^\top \mathbf{P}_e V_e$
Assemble global force $\mathbf{f}_{\text{int}} = \sum_e \mathbf{f}_{\text{int}}^e$
// Neural integration module
$\Delta\mathbf{v} = g_\phi(\mathbf{x}_t, \mathbf{v}_t, \mathbf{f}_{\text{int}})$
$\mathbf{v}_{t+1} = \mathbf{v}_t + \Delta\mathbf{v}$
$\mathbf{x}_{t+1} = \mathbf{x}_t + h\mathbf{v}_{t+1}$
$\mathbf{x}_{t+1}, \mathbf{v}_{t+1} = \mathcal{C}(\mathbf{S}_{BC}(\mathbf{x}_{t+1}, \mathbf{v}_{t+1}))$

Neural constitutive module In physics simulation, internal forces $\mathbf{f}_{\text{int}}$ are determined by the material's constitutive law through the following process. First, the strain energy density function $\Psi(\mathbf{F})$ defines the material's response to deformation, where $\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}$ is the deformation gradient measuring local geometric distortion from reference coordinates $\mathbf{X}$ to deformed coordinates $\mathbf{x}$. Then, the first Piola-Kirchhoff stress $\mathbf{P}$ is computed as $\mathbf{P} = \frac{\partial \Psi(\mathbf{F})}{\partial \mathbf{F}}$, representing the material's resistance to deformation. Internal forces are calculated by adding up all elements: $\mathbf{f}_{\text{int}} = \sum_e \mathbf{B}_e^\top \mathbf{P} V_e$, where $\mathbf{B}_e$ is the strain-displacement matrix for element e, V_e is the element's volume in the reference configuration.

In this module, the classical computation flow is maintained except that the computation of the stress term $\mathbf{P}$ is replaced by a learnable neural network f_θ, since the material response is usually hard to measure in real-world scenarios, where the data-driven paradigm is more flexible. Here, f_θ is configured as a special rotation equivariant architecture following [22]. Concretely, given $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, we feed the rotation-invariant tuple ($\mathbf{\Sigma}$, $\mathbf{F}^\top\mathbf{F}$, $\det \mathbf{F}$) into a two-layer MLP, then rotate the output back with $\mathbf{R} = \mathbf{U}\mathbf{V}^\top$, yielding $\mathbf{P} = \mathbf{R}\, f_{\boldsymbol{\theta}}(\cdot)$. This modularization maintains FEM's element-wise computation flow while leveraging learning to capture complex material behaviors that traditional models may fail to capture in real-world applications.

Notably, the parameterization of f_θ adopted from [22] is only a part of our thorough modular framework, which targets a fully modular neural simulator and introduces a more comprehensive methodology to address challenges in physics modularization and model

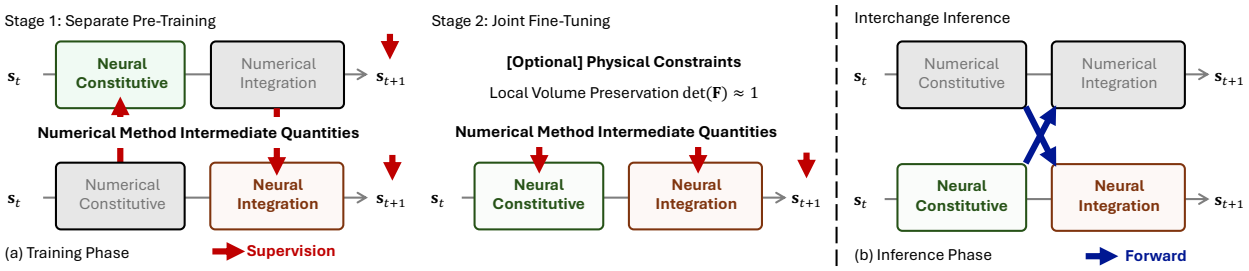


Figure 5.2: Modular training strategy and inference scheme. A two-stage training paradigm is used to optimize neural modules. Separate training in the first stage prevents module collapse, while joint fine-tuning in the second stage further leverages physical constraints.

optimization.

Neural integration module In physics simulation, *e.g.*, FEM, the equations of motion use various time integration schemes, most commonly explicit and implicit Euler:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + h\mathbf{v}_{t+\alpha}, \;\; \mathbf{v}_{t+1} = \mathbf{v}_t + h\mathbf{M}^{-1}\left(\mathbf{f}_{\text{int}}(\mathbf{x}_{t+\alpha}) + \mathbf{f}_{\text{ext}}\right), \tag{5.1}$$

where h is the timestep and the choice of α selects an integration scheme: $\alpha = 0$ (explicit) and $\alpha = 1$ (implicit). $\mathbf{M}$ denotes the (lumped) mass matrix and $\mathbf{f}_{\text{ext}}$ represents the external force, which are both hard to access in the real world. For example, it is hard to decide the mass matrix $\mathbf{M}$ since the object can be hollow or solid and the environment's frictional forces depend on the contact material. This motivates us to compute $\mathbf{M}$ and $\mathbf{f}_{\text{ext}}$ related terms in a learnable way.

Specifically, the neural integration module preserves the same interface as an implicit numerical method, but approximates velocity increment $\Delta\mathbf{v}$ ($\mathbf{M}, \mathbf{f}_{\text{ext}}$-related term in Eq. 5.1) with a learnable neural network:

$$\Delta\mathbf{v} = g_\phi(\mathbf{x}_t, \mathbf{v}_t, \mathbf{f}_{\text{int}}) \quad \text{with} \quad \begin{aligned} \mathbf{v}_{t+1} &= \mathbf{v}_t + \Delta\mathbf{v} \\ \mathbf{x}_{t+1} &= \mathbf{x}_t + h\mathbf{v}_{t+1} \end{aligned} \tag{5.2}$$

For the configuration of g_ϕ, a major obstacle is that the raw coordinates $\mathbf{x}_t$ vary by global translation and rotation between scenes and simulation steps. We therefore prepend a learnable *canonicalisation* module: a two-layer T-Net [104] predicts a 3×3 linear map $\mathbf{T}$ that aligns the deformed mesh to a learned reference frame, $\tilde{\mathbf{x}}_t = \mathbf{T}(\mathbf{x}_t - \bar{\mathbf{x}}_t)$, where $\bar{\mathbf{x}}_t$ is the mesh centroid. Then, we embed aligned positions, original positions, velocities and internal forces of each vertex with MLP layers; concatenating these embeddings with the raw inputs yields a compact feature vector describing the mesh state. Afterwards, another MLP then maps the feature to the velocity increment $\Delta\mathbf{v} \in \mathbb{R}^3$.

Beyond elaborative physical alignment, the above modularization has several favorable properties. First, it exposes intermediate quantities at module interfaces, which can be directly supervised by intermediate quantities of numerical methods during training, further boosting the model interpretability and module specialization. Second, neural modules maintain the same interface as traditional simulators, allowing interchange with their traditional physics-based counterparts for flexible deployment (Fig. 5.2b).

5.2.3 Modular Physics Training

As demonstrated by [73], a modular architecture is not enough to guarantee that the whole model works modularly; the training strategy is also essential. Therefore, as presented in Fig. 5.2, we present a two-stage training strategy to optimize the predefined neural modules.

Separate training In the first phase, we train the neural constitutive module and neural integration module independently, along with the differentiable simulation counterpart for the other component. Specifically, for each timestep, the supervision can be formalized as follows:

$$\begin{aligned} f_\theta\text{: } \mathcal{L}_{\text{constitutive}} &= \|\mathbf{f}_{\text{int}} - \mathbf{f}_{\text{int}}^*\|_2^2 + \|\mathbf{x}_{\text{ref-integration}} - \mathbf{x}^*\|_2^2 \\ g_\phi\text{: } \mathcal{L}_{\text{integration}} &= \|\mathbf{v} - \mathbf{v}^*\|_2^2 + \|\mathbf{x}_{\text{ref-constitutive}} - \mathbf{x}^*\|_2^2, \end{aligned} \tag{5.3}$$

We omit timestep subscript for simplification. $\mathbf{x}_{\text{ref-}}$ represents the result generated with reference numerical component. $\cdot^*$ represents the supervision generated by numerical methods. Benefiting from the unique tractable property of numerical methods, we can also enable direct supervision for the intermediate results of neural modules, ensuring each network learns its specific role, namely, material behavior or temporal dynamics. Additionally, we also utilize the differential numerical modules along with neural counterparts to generate simulation results, where the supervision on the final result $\mathbf{x}$ can serve as a regularization term to ensure each module's output is physically meaningful.

Joint finetuning After separate training, we obtain two well-optimized modules, which are then jointly fine-tuned to adapt to each other while preserving their physical understanding from pretraining. Notably, by exposing intermediate physical quantities such as deformation gradients, NMP enables analytical computation of physical constraints, which was previously infeasible in monolithic neural simulators due to their opacity.

To demonstrate our framework's flexibility in physical constraining, we further explore the incorporation of *local volume preservation* of elastic dynamics into the loss function to improve the physical consistency of simulations. Specifically, this preservation corresponds to the fact that, for nearly incompressible elastomers, the determinant of the deformation gradient, $\det(\mathbf{F})$, should stay close to 1. Thus, we additionally penalize excess deviation with a hinge loss [105]:

$$\mathcal{L}_{\text{volume}} = \sum\nolimits_e \left[\max\left(|\det(\mathbf{F}_e) - 1| - \epsilon_{\text{volume}},\, 0\right)\right]^2, \tag{5.4}$$

where e indexes elements and we set $\epsilon_{\text{volume}} = 0.05$ following engineering practice. During finetuning, this newly added penalty is added to the L2 loss for intermediate physical quantities and model predictions with a weight of 0.1. In experiments, this physical constraint will not drastically improve the accuracy metric but can consistently enhance the physical soundness of model predictions.

Note that the above physical constraint on local volume preservation is only an example. With our modular physics framework, it is flexible to add new physical regularization to the training process, such as the L2 distance between the elastic potential energy $\sum_e V_e\, W_e(\mathbf{F}_{e,t})$ of the neural and numerical simulators, where V_e and $W_e(\cdot)$ represent rest volumes and strain-energy density respectively.

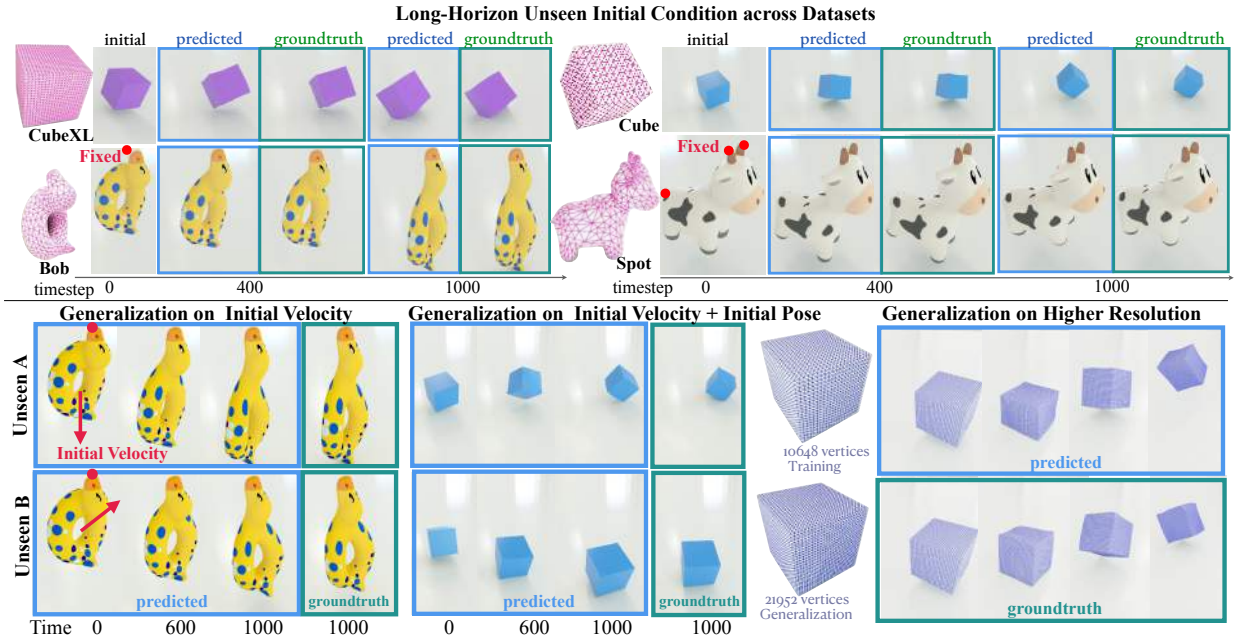


Figure 5.3: Long-Horizon Stability and Generalization of Neural Modular Physics. Top: Long-horizon rollouts under random initial conditions, showing the rest mesh and simulation keyframes on all datasets. Ours remains stable and closely matches the ground truth (FEM) over 1,000 steps. Bottom: Generalization to unseen initial conditions and higher resolutions, including unseen initial velocity (BOB), unseen velocity and pose (CUBE), and resolution generalization (CUBEXL, 10k→21k vertices).

5.3 Experiments

We construct five test environments to evaluate different aspects of elastic simulation. The first four environments are designed with complete physical information to test a model's capacity for physics learning, while the final environment introduces unknown friction to imitate a real-world challenge.

CUBE We simulate an elastic cube (1,000 vertices, 3,645 tetrahedra) dropping and bouncing on a rigid floor under gravity. This scene tests external collision handling and general deformation behavior. The validation set tests generalization by varying initial velocity and cube orientation, examining both collision response and general deformation behavior under large dynamics.

CUBEXL A high-resolution cube (22^3 grid, 10,648 vertices) to stress-test the model on large-resolution meshes.

SPOT The little cow "Spot" [106] (1,015 vertices, 2,752 tetrahedra) is constrained at both head and tail. This environment tests the handling of multiple fixed boundary points and stress propagation in complex geometry. The validation set varies initial velocities.

BOB An elastic duck "Bob" [107] (849 vertices, 3,108 tetrahedra) is suspended in air with head fixed. This environment tests the method's ability to handle Dirichlet boundary conditions at fixed points and complex geometry deformation. The validation dataset varies the initial velocity.

For these four environments, we use a neo-Hookean material model [108] with timestep

Table 5.1: RMSE between model-simulated vertex positions and ground-truth simulations across rollout horizons. See Section 5.3.6 for baseline visualizations.

Method	Cube			CubeXL			Spot			Bob		
	100	500	1000	100	500	1000	100	500	1000	100	500	1000
Transolver	0.277	1.525	3.200	0.362	3.622	9.984	0.229	0.943	2.290	0.075	0.597	1.782
EGNO	0.503	2.308	4.492	0.662	3.288	6.357	0.370	1.176	1.217	0.299	1.414	2.770
MGN	0.151	0.371	1.554	0.082	0.433	1.963	0.066	0.577	2.364	**0.048**	0.508	2.104
Ours	**0.019**	**0.159**	**0.535**	**0.020**	**0.253**	**0.440**	**0.061**	**0.217**	**0.449**	0.050	**0.316**	**0.579**

$h = 0.0005$ seconds, computed via a semi-implicit integrator. Each dataset has 32 training trajectories of 1,000 steps, with randomly initialization, plus 8 validation trajectories with unseen initial conditions. Additional details are provided in Section 5.4.

Friction The final benchmark introduces unknown friction dynamics. An elastic rubber duck "Bob" [107] (5,130 vertices, 21,448 tetrahedra) is dropped onto a rough surface and begin sliding under frictional contact dynamics. Trajectories are generated with an external Projective Dynamics simulator [39], which models both hyperelastic material response and frictional contact through an implicit integration scheme. Unlike the previous environments where contact reduces to simple non-penetration constraints, this setting requires the neural network to capture rich frictional effects that are not provided explicitly. This benchmark therefore evaluates the model's ability to generalize to more realistic contact phenomena and infer missing dynamics directly from data.

5.3.1 Main Results

Baselines As discussed in the introduction, NMP is a fully neural modular framework. We therefore compare against three representative neural simulators: the GNN-based MeshGraphNet (MGN) [19], the equivariant neural operator EGNO [109], and the Transformer-based Transolver [21]. Pure physics and hybrid neural–physics simulators are also evaluated in the next section.

Setups We evaluate all methods on 100, 500, and 1,000-step rollouts from unseen test trajectories across all environments, assessing long-horizon stability and error accumulation. During training, models are supervised on sequences of at most 200 steps, making the 500- and 1,000-step rollouts a temporal extrapolation.

Results As shown in Table 5.1, compared with other neural simulators, NMP achieves much better accuracy with over 60% error reduction in the 1,000-step rollout, highlighting the benefits of modular design in capturing long-horizon dynamics Despite our careful tuning, advanced neural operators such as EGNO and Transolver still fail in long-term simulation. This may be that without explicit supervision on intermediate physics quantities, these monolithic neural simulators are very likely to generate meaningless results, especially under long-term rollout. See Section 5.3.6 for visual comparison, Section 5.3.9 for statistics of per-sequence standard deviations of Table 5.1, and Section 5.4.4 for implementation details.

Visualizations As shown in Fig. 5.3 top, although NMP is trained on sequences of at

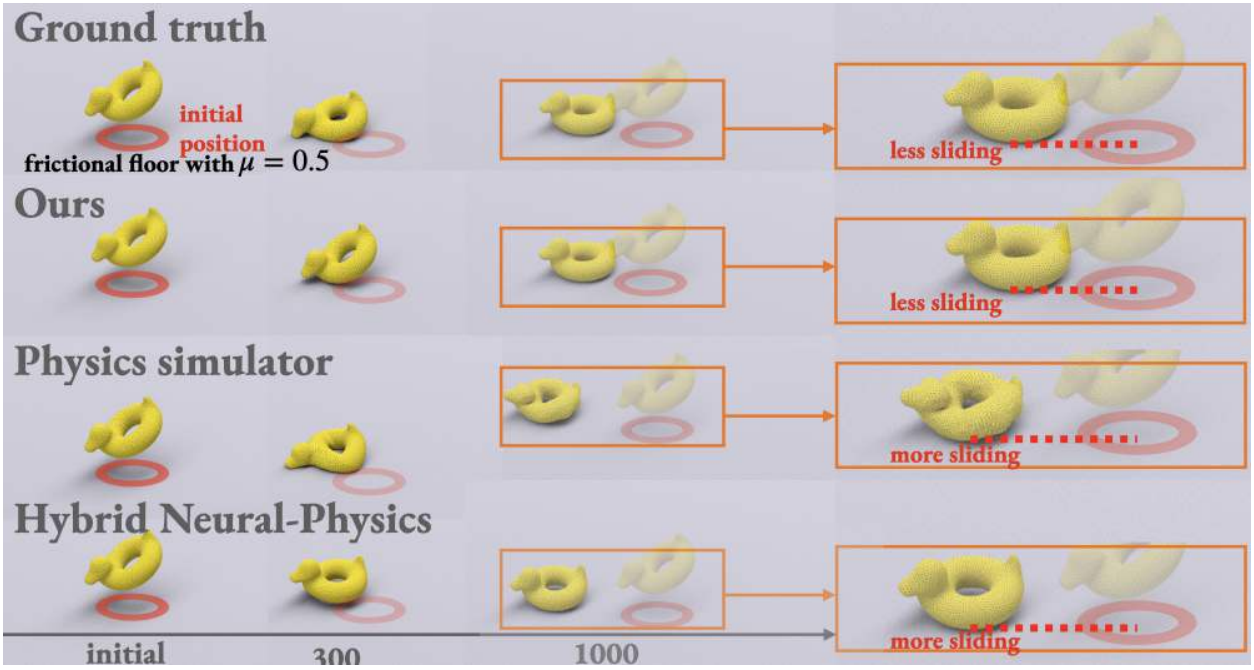


Figure 5.4: Simulation comparison on FRICTION environment with unknown frictional dynamics. We mark the initial position of duck-surface contact with red circles.

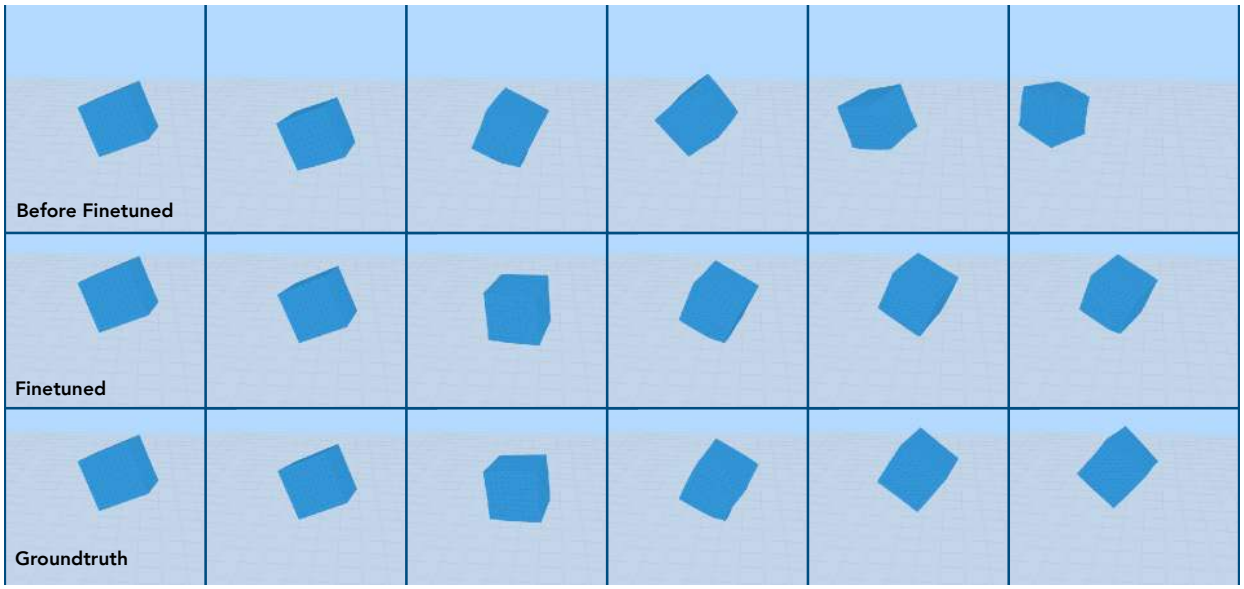


Figure 5.5: Sim-to-real proof of concept. We pretrain an NMP model on CUBE and finetune it on a scenario with unknown external force and a softer material. To mimic the real-world simulation task, only the position information is accessible during finetuning.

Table 5.2: Inference efficiency comparison on BOB for a 1,000-step rollout.

Method	Time (ms)
Transolver [2024]	11796
EGNO [2024]	26351
MGN [2021]	27603
NCLaw [2023]	97121
Physics Simulator	90275
NMP (Ours)	**2250**

most 200 steps, it remains stable and physically plausible over extended simulations. In contrast, monolithic baselines typically diverge after a few hundred steps (Section 5.3.6). These results show the efficacy of our design in supporting stable extrapolation.

Unseen initial conditions generalization We evaluate two unseen initial conditions—novel velocities and combined velocity + pose offsets—for BOB and CUBE (Fig. 5.3 bottom left). Our modular simulator adapts to both, yielding stable, plausible 1,000-step trajectories that closely track ground truth despite no such examples in training. Results for SPOT and CUBEXL are provided in Section 5.3.8.

Higher resolution generalization We test mesh-agnostic generalization by evaluating on a finer discretization than seen in training ($10k \rightarrow 21k$ vertices; Fig. 5.3 bottom right). Despite being trained at a single coarse resolution, our per-vertex module design scales seamlessly to denser meshes, maintaining stable and physically accurate motion.

5.3.2 Comparison with Physics Simulators

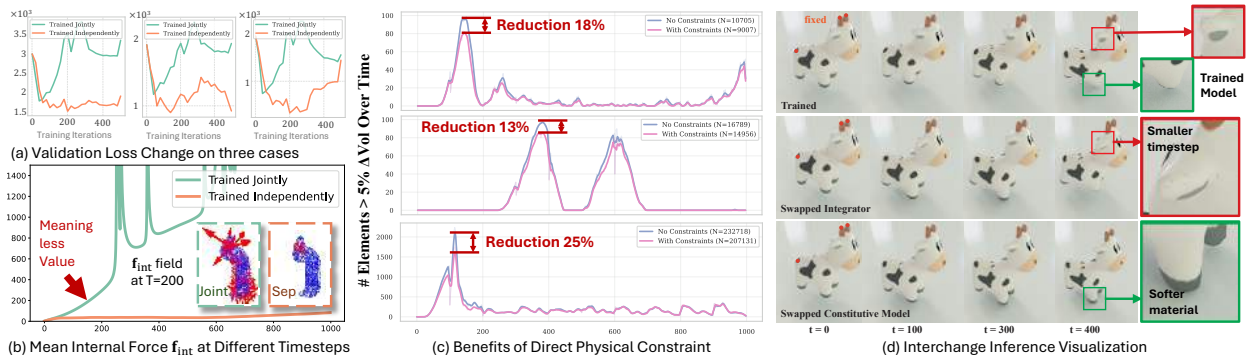


Figure 5.6: (**a-b**) Comparison between joint training two modular networks and our two-stage independent training strategy in BOB. **(c)** *Impact of physical constraints* on three trajectories in CUBE. The curve records the number of tetrahedra exceeding 5% volume change at each time step and the legend counts the total number violating during 1,000 steps. **(d)** *Interchange inference.* Top: Sequence generated by our trained base model. Middle: Replacing the neural integration module with a semi-implicit integrator with a much smaller timestep. Bottom: Replacing the neural constitutive module with a softer St. Venant material increases deformation, visible in the foot position.

This section presents a comparison between NMP and physics simulators. Specifically, our baselines include *(i)* a traditional physics simulator and *(ii)* the hybrid neural-physics

method NCLaw [22] that augments a numerical integrator with a learned material model.

Scenario flexibility In the FRICTION environment, an elastic duck interacts with a surface through unknown frictional dynamics, and all methods are evaluated on unseen 1,000-step rollouts. Because neither *pure* nor *hybrid* physics simulators have access to the underlying contact law, they cannot capture these frictional effects; we therefore perform system identification to estimate reasonable input parameters for these baselines. In contrast, our model learns directly from data and does not require system identification.

As shown in Fig. 5.4, our approach learns the frictional behavior, with the duck stopping after a short slide rather than gliding without resistance, indicating successful learning of the unknown contact dynamics. Although NCLaw captures material response, both NCLaw and the pure physics simulator fail to model the contact interactions. These results demonstrate that our neural simulator offers superior scenario flexibility under incomplete physical information, highlighting the advantage of neural approaches in handling unknown dynamics where numerical simulators are limited.

Efficiency comparison In addition to scenario flexibility, another advantage of neural simulators is their efficiency [21, 68]. Here we also measure the running efficiency of various methods. As listed in Table 5.2, all neural baselines and our method exhibit significantly better efficiency than physics and neural-physics hybrid methods. This efficiency arises from the forward-only inference of neural simulators and the highly optimized tensor operations on modern hardware [95]. In particular, NMP utilizes a specialized modularization architecture, which allows us to introduce only lightweight neural modules, rather than the unwieldy monolithic model. Therefore, NMP is around 40× faster than the traditional physics simulator and over 5× faster than the advanced neural operator Transolver [21], where the latter has to infer the complete unwieldy model at each step.

Transferability Since NMP employs a two-stage training strategy, the internal force is required (Eq. 5.3) during training to ensure a successful modularization. Such intermediate physics is accessible in the simulation scenario but cannot be recorded in the real world. Here, we demonstrate that NMP can also be applied to real-world simulation by utilizing the transferability of neural networks.

Here we consider a new scenario, where a cube drops and bounces under an unknown external force and unknown material (softer than the CUBE dataset) and only position is accessible. Since such a partially observable simulation cannot provide intermediate physics for two-stage training, we adopt the pretrain-finetune paradigm of deep learning and pretrain NMP on the simulated CUBE dataset for a well-modularized model, then finetune it solely based on the position information. Figure 5.5 shows that NMP can still generate simulations accurately. This result not only justifies the practicality of NMP but also highlights the advantage of neural simulators in transferability.

5.3.3 Unique Benefits of Modularization

Notably, it is not easy to guarantee that a complex neural modular network works as we expected. Without the elaborative physical-aligned architecture and specialized training strategy in NMP, we cannot realize the unique benefits of modularization. To provide an intuitive understanding, we compare our two-stage approach against joint training of both networks from scratch in Fig. 5.6(a-b). It can be observed that training NMP with

two modular networks jointly will make the interface quantity $\mathbf{f}_{\text{int}}$ physically meaningless, which presents a noisy force field with impossibly large values. This confused intermediate representation indicates the modular network does not work as we expected, namely, the collapse issue [73]. In contrast, our special training strategy can help the modular network learn realistic physical quantities and stabilize the training process. As a result, the properly specialized modular network empowers the model with interesting features.

Direct physical constraint As stated before, our modularization maintains strict physical alignment with traditional simulators, enabling seamless physical constraint (Eq. 5.4). This feature cannot be accomplished in previous monolithic simulators. To further present the benefit brought by incorporating physical constraints, we compare our full method against a variant trained without the volume preservation losses, where the number of tetrahedral elements that exceed 5% volume change during simulation is recorded. As presented in Fig. 5.6(a), the model trained with local volume preservation shows better physical soundness. Specifically, the loss can further reduce the maximum number of violations by 10%. Despite the simulation of NMP still being imperfect, the special modularization provides us with an interface to explicitly constrain physical rules.

Interchange inference Benefitting from our specialized design of modular interfaces, NMP can seamlessly achieve interchange inference with traditional simulators. To verify the interchange stability, we first swap the neural integrator for a semi-implicit numerical integrator with a tiny timestep (5×10^{-6}), $100\times$ smaller than during training (Fig. 5.6(b), middle). This captures higher-frequency dynamics, such as Spot's ear motion, showing that our learned constitutive module remains compatible with alternative numerical integrators. Second, we replace the neural constitutive model with a softer St. Venant material (Young's modulus 5×10^4), producing greater deformation (Fig. 5.6(b), bottom). These experiments highlight our framework's plug-and-play flexibility across integrators and materials, enabling fast adaptation to new behaviors without retraining.

5.3.4 Ablation Study

Here, we include some ablations about the hyperparameters and configurations of neural modules.

Model complexity analysis We ablate the size of the neural constitutive model to assess sensitivity to network complexity. Our original architecture (2 hidden layers, 96 and 80 neurons) was chosen to maximize GPU memory usage during training. Reducing the architecture to 2 layers of 64 and 40 neurons, and evaluating on the Cube dataset with a ground-truth integrator, we observe a higher 1000-step RMSE (0.337 ± 0.270) compared to the original model (0.220 ± 0.174), highlighting the importance of model size. One promising direction is to further scale up the NMP model.

Hyperparameter analysis As stated in Eq. 5.4, adding physical constraint also introduces a weight hyperparameter. In the experiments, we selected the volume regularization constant such that the constraint term contributed approximately 1% to the total loss during training. This choice reflects a balance between encouraging physical plausibility and preserving

Table 5.3: Physical constraint loss weight analysis. The following test is conducted on the "real-world" CUBE scenario in Figure 5.5.

Loss Weight	1000-step RMSE
0	0.397
1	0.337
10	0.604
1000	2.579

learnability. Here, we evaluated model performance under a range of volume loss weights $\{0, 1, 10, 1000\}$. As shown right in Table 5.3, moderate regularization (e.g., 1.0) improves long-horizon accuracy, while excessively large values (e.g., 1000) degrade performance because large weight would discourage mesh achieving desired deformation.

5.3.5 Failure Mode Analysis

To further demonstrate the simulation property of our model, we select the worst case (with the largest RMSE) among the BOB task. The visual comparison is included in Figure 5.7. It is observed that even in the worst case, NMP still accurately simulates the main dynamics of Bob. Besides, we can also notice that dynamics with large deformation are really challenging for simulation.

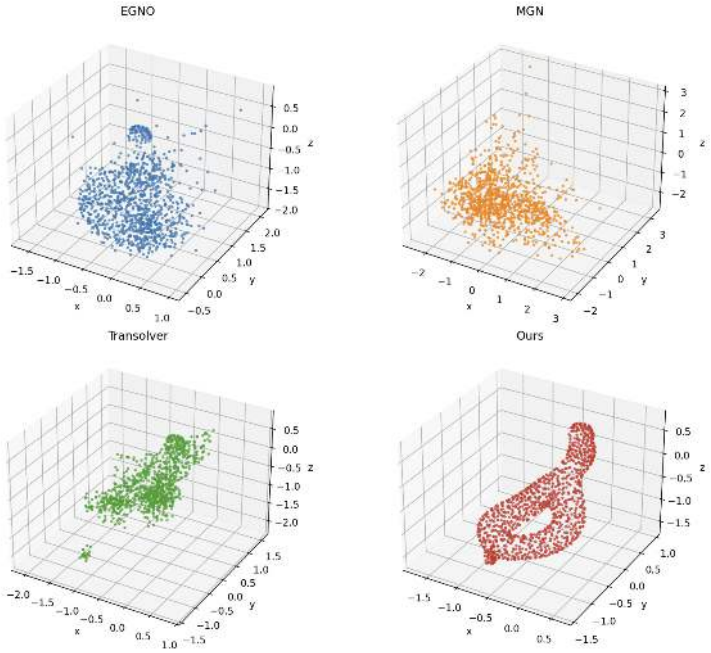


Figure 5.7: Failure mode analysis. We visualize the last frame in the worst case, which is selected according to the RMSE of NMP. The predictions of other baselines are also included.

Figure 5.8: Qualitative comparison of rollouts for the finite element ground truth and baselines at 100, 500, and 1000 steps on Bob. From top to bottom, rows show Ground Truth, MeshGraphNet, EGNO, and Transolver. Columns show snapshots at 100, 500, and 1000 steps.

5.3.6 Comparison with Baselines

Visual comparison As presented in Fig. 5.8, we compare Transolver [21], EGNO [109], and MeshGraphNet (MGN) [19] against the ground truth at 100, 500, and 1000 steps. The visualizations reveal several distinct patterns in the baselines' performance.

Among the three neural simulators, Transolver best preserves geometry at short horizons. At 100 steps, its simulated shape and pose remain close to the reference, and by 500 steps, the results still appear coherent, following the trajectory with only moderate drift. However, by 1000 steps, Transolver loses local structure, leaving only a rough silhouette that reflects cumulative error and the loss of elastic memory.

EGNO deteriorates much earlier. By 500 steps, the configuration has already sagged into a blurred particle cloud with a strong downward bias, and by 1000 steps, the object becomes largely amorphous, with the original geometry no longer discernible.

MeshGraphNet maintains a recognizable outline longer than EGNO, but its motion is strongly overdamped. Oscillations quickly fade, contact responses smear, and the particles exhibit a spurious sinking or condensing trend rather than elastic rebound. By 1000 steps,

Table 5.4: Compare with enhanced baselines, where each model will directly predict 100 steps at one inference. The RMSE of 1,000 steps is recorded.

Method (Bob simulation)	RMSE
Transolver [2024] (predict 100-steps)	0.75
EGNO [2024] (predict 100-steps)	2.00
MeshGraphNet [2021] (predict 100-steps)	2.61
NMP (Ours, predict one-step)	**0.58**

the geometry stretches along the vertical direction and global coherence breaks down.

These results demonstrate the difficulty for monolithic neural simulators to accurately capture long-horizon physical dynamics. In contrast, as shown in Fig. 5.3, our proposed neural modular physics framework is able to generate stable and physically consistent simulations over extended horizons.

New autoregressive prediction In neural simulator baselines, their official configuration is to adopt the one-step autoregressive prediction. Thus, we also follow this setting in our experiments for both our method and baselines. One possible way to enhance these baselines is to make the model predict 100 steps at once; then the 1,000-step simulation only requires 10 times of rollout prediction, which is a practical trick to reduce the accumulation error.

Here, we also compare with enhanced baselines. As listed in Table 5.4, directly predicting 100 steps can reduce the long-term simulation error of neural simulators, especially Transolver, whose RMSE is reduced from 1.782 (Table 5.1) to 0.75. However, these baselines still fall behind our method, which highlights the advancement of modularization.

5.3.7 Initial Condition Distribution Illustration

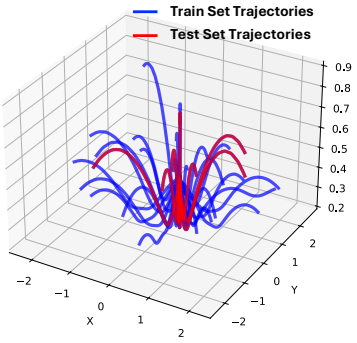


Figure 5.9: Visualization of trajectories with different initial conditions on Cube.

As stated in the main text, different simulation trajectories begin with distinct initial conditions, which naturally lead to divergent dynamic trajectories. To provide an intuitive illustration of this effect, we plot 40 trajectories from the Cube dataset, including 32 cases from the training set and 8 cases from the test set. In these cases, variations arise from different initial velocities and cube orientations. As shown in Fig. 5.9, even small differences

in initial velocity or orientation at the first step can significantly alter subsequent dynamics, creating a pronounced gap between training and test trajectories. This visualization highlights the inherent challenge of generalization in our experiments.

5.3.8 Generalization to Unseen Initial Conditions

Fig. 5.10 shows our method's generalization capability on the Spot scene across different initial conditions. Similar to the Bob scene discussed in the main text, we evaluated the model on multiple unseen initial velocities. The results demonstrate consistent performance across different scenarios - each row shows a unique trajectory where the model maintains stable motion while accurately preserving the fixed-point boundary conditions at both head and tail. At $t = 1000$, our predictions (bounded by blue boxes) closely match the ground truth configurations (within green boxes), indicating robust generalization despite the challenging dual-fixed-point constraints.

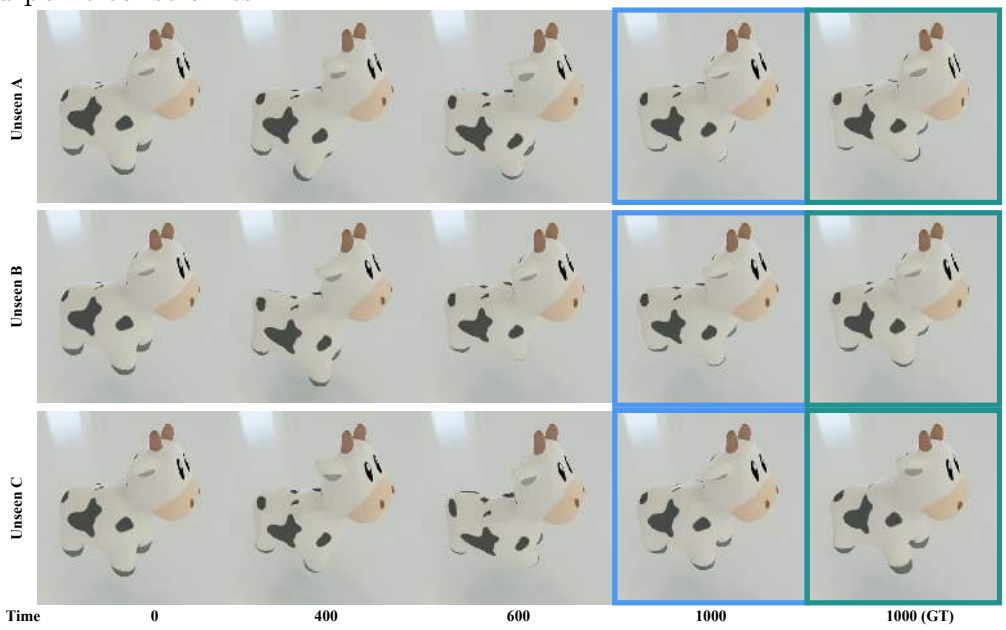


Figure 5.10: Generalization to unseen initial conditions for Spot. Each row shows a sequence with different initial velocities, demonstrating our model's ability to handle varied dynamics while maintaining fixed-point constraints. Our predictions (blue) closely match the ground truth (green).

5.3.9 Standard Deviations for Table 5.1

Table 5.5 reports the per-sequence standard deviations (over 8 validation trajectories) corresponding to the performance means listed in Table 5.1.

5.3.10 Physics Validation Metrics

In addition to RMSE, the physical soundness of simulators is also an essential metric. Therefore, as a supplement to Figure 5.6, we also count the number of tetrahedra exceeding 5% volume change during a 1,000-step rollout, which measures the local volume preservation

Table 5.5: Standard deviations of RMSE for Table 5.1 (computed over 8 validation trajectories).

Method	Cube			Cube XL			Spot			Bob		
	100	500	1000	100	500	1000	100	500	1000	100	500	1000
Transolver	±0.069	±0.364	±0.706	±0.073	±0.726	±2.323	±0.062	±0.343	±0.673	±0.043	±0.230	±0.563
EGNO	±0.197	±0.979	±2.001	±0.103	±0.815	±1.533	±0.098	±0.333	±0.273	±0.110	±0.524	±1.038
MeshGraphNet	±0.002	±0.009	±0.002	±0.010	±0.034	±0.631	±0.001	±0.003	±0.001	±0.001	±0.001	±0.002
Ours	±0.010	±0.046	±0.161	±0.010	±0.149	±0.107	±0.001	±0.135	±0.162	±0.021	±0.136	±0.348

Table 5.6: Physics validation of different models, where we record the number of tetrahedra exceeding 5% volume change. A smaller number indicates better physical soundness.

Method (Cube simulation)	Number
Transolver [2024]	1271827
EGNO [2024]	1595317
MeshGraphNet [2021]	479566
NMP (Ours)	**231094**

of different simulators. As shown in Table 5.6, NMP surpasses all the other baselines in this physics validation metric. These results further demonstrate the benefits of neural modular physics in guaranteeing physical soundness.

5.4 Implementation Details

In this section, we will describe more details of our experiments in the main text.

5.4.1 Groundtruth Simulator

We implemented our ground truth simulator in PyTorch to enable end-to-end differentiability. The simulator employs a semi-implicit integration scheme:

$$\mathbf{v}_{t+1} = \mathbf{v}_t + dt(\mathbf{M}^{-1}\mathbf{f} + \mathbf{g}) \tag{5.5}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + dt\mathbf{v}_{t+1}, \tag{5.6}$$

where $dt = 5 \times 10^{-4}$ seconds denotes the timestep, $\mathbf{M}^{-1}$ represents the inverse mass matrix, $\mathbf{g}$ is gravity acceleration that is set to $-9.8m/s^2$ in our experiments.

After integration, the simulator handles external constraints. For scenes involving ground contact (e.g., Cube scene), vertices that fall below the ground plane are projected back to ground level with their vertical velocities set to zero. For scenes with fixed boundary conditions (Bob and Spot), designated vertices are maintained at their rest positions with zero velocity throughout the simulation.

5.4.2 Dataset Generation

Timestep setting For the first four environments, ground truth trajectories are generated with a small internal timestep of $dt = 10^{-5}$ seconds (50 substeps per output frame) to ensure

stability. Each trajectory contains 1000 frames at a coarser resolution of $dt = 5 \times 10^{-4}$, which corresponds to one output frame per 50 simulation steps. Importantly, our model is only trained and evaluated on these coarser frames, meaning the neural integrator operates at the larger effective timestep of $dt = 0.0005$. For the FRICTION environment, we generate and test the simulator at $dt = 0.0001$ to test a larger timestep.

The above setup inherently challenges the model to predict dynamics over long time intervals, encouraging temporal generalization. While training uses rollouts of up to 200 steps, all reported results in the main text, including the 1000-step predictions, require the model to generalize far beyond its training horizon.

Material configuration For dataset generation of the first four environments, we used the neo-Hookean material model:

$$W_{NH}(\mathbf{F}) = \frac{\mu}{2}(I_C - 3) - \mu \ln(J) + \frac{\lambda}{2}(\ln(J))^2, \tag{5.7}$$

where μ and λ are Lamé parameters, $J = \det(\mathbf{F})$ is the volume change ratio, and $I_C = \mathrm{tr}(\mathbf{F}^T\mathbf{F})$ is the first invariant of the right Cauchy-Green deformation tensor.

Scene-specific material parameters were configured as follows. We configured different material stiffness across scenes while maintaining the same Poisson's ratio $\nu = 0.45$. The cube scene uses a stiffer material with Young's modulus $E = 5 \times 10^5$ Pa, while both Bob and Spot scenes use a more compliant material with $E = 1 \times 10^5$ Pa to allow for larger deformations.

For the FRICTION environment, we used the differentiable simulator DiffPD [39], which is based on projective dynamics and thus requires a projective dynamics formulation of material energies. We refer the reader to the original paper for full details of the energy discretization and solver. For consistency with the other environments, we set the material parameters in the same manner as above, maintaining Poisson's ratio $\nu = 0.45$ while varying the Young's modulus E to control material stiffness.

5.4.3 Network architecture and training details

Model architecture The neural constitutive module is implemented as a two-layer MLP with 80 and 96 neurons, respectively. Our neural integration module predicts per-vertex velocity updates $\Delta\mathbf{v}$ given positions $\mathbf{x}$, velocities $\mathbf{v}$, and internal forces $\mathbf{f}_{\text{int}}$. Each of the three inputs is processed independently by a one-layer MLP with SiLU activation, producing embeddings of 32 dimensions respectively. Additionally, we use a learned alignment module (TNET) to produce aligned input mesh coordinates: it outputs a 3×3 rotation matrix (via a quaternion regressor) that aligns each mesh to a canonical frame. This promotes pose invariance and improves generalization to varied initial conditions. The aligned mesh positions, original mesh positions, velocities and forces are concatenated with the original inputs and passed through a three-layer MLP with 64 hidden units to regress $\Delta\mathbf{v}$. To regularize early training, we clamp predicted velocities to a maximum of 30 m/s in magnitude, following MeshGraphNet [19].

Learned alignment module To promote generalization across varying initial poses, we use a learned spatial alignment module that normalizes input positions by predicting a global

3×3 linear transformation matrix per sample. The input point cloud $\mathbf{x} \in \mathbb{R}^{N\times 3}$ is first encoded by a shared MLPNET applied independently to each vertex. The resulting per-point features are aggregated via max pooling, then passed through a two-layer MLP to regress a 9-dimensional vector, reshaped into a 3×3 matrix. To encourage stability, the predicted matrix is initialized near the identity. Unlike prior approaches that constrain the output to be a rotation (e.g., via quaternions or projection to SO(3)), we allow a full linear transformation, enabling the network to learn scale, shear, and other canonicalizing deformations directly from data.

Training configurations Both networks are trained using the Adam optimizer with a cosine annealing learning rate schedule. The training process consists of two phases: independent pretraining of each network for 100 epochs, followed by joint fine-tuning for another 100 epochs. During pretraining, each network is trained with ground truth inputs from the simulation dataset, while fine-tuning allows the networks to adapt to each other's learned behaviors.

To stabilize sequence training and prevent error accumulation, we employ a teacher forcing strategy during both pretraining and fine-tuning. When processing sequences within each epoch, the simulation state is periodically reset to ground truth values. The reset interval increases from 60 to 200 steps following a cosine annealing schedule, gradually encouraging the networks to handle longer sequences of predictions without intervention.

5.4.4 Baseline Implementation

We implement three representative neural simulators, Transolver [21], EGNO [109], and MeshGraphNet (MGN) [19]. To ensure fairness and reproducibility, all models are built on *their official codebases or libraries* and follow the recommended preprocessing and data pipelines where applicable. All baselines are optimized with the normalized mean squared error (NMSE) loss $\mathrm{NMSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\|\hat{\mathbf{y}}-\mathbf{y}\|_2^2}{\|\mathbf{y}\|_2^2}$, using a fixed learning rate of 0.1 and training for 100 epochs. The following are detailed configurations:

- **Transolver** [21]: we adopt the Transolver Irregular Mesh variant and use 15 Transolver blocks with model width 256, 4 attention heads, an MLP ratio of 2, and dropout 0.1, while setting the slice number to 32 and the number of reference tokens to 16.
- **EGNO** [109]: 15 layers with hidden size 256, 4 vector-message heads, dropout 0.1, and an RBF embedding of dimension 12.
- **MeshGraphNet** [19]: 15 message-passing steps with hidden size 128.

To improve robustness and stabilize normalization statistics, we inject small Gaussian noise into the inputs of *all* three neural simulators (Transolver, EGNO, and MeshGraphNet) during training. Concretely, at each step, we perturb positions and velocities as $x_t \leftarrow x_t + \varepsilon_x$, $v_t \leftarrow v_t + \varepsilon_v$ with $\varepsilon_x, \varepsilon_v \sim \mathcal{N}(0, \sigma^2)$, where we use $\sigma = 3 \times 10^{-3}$.

For the hybrid physics simulator, we directly follow their official configurations for the model architecture configuration of NCLaw [22]. For the physics simulator baseline, we use our groundtruth simulator described in Sec. 5.4.1.

5.4.5 Interchange Inference Experiment Settings

For material model swapping experiments, we used the St. Venant-Kirchhoff model:

$$W_{StVK}(\mathbf{F}) = \frac{\lambda}{2}\left(\mathrm{tr}(\mathbf{E})\right)^2 + \mu \mathrm{tr}(\mathbf{E}^2), \tag{5.8}$$

where $\mathbf{E} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I})$ is the Green strain tensor.

5.5 Discussion and Conclusion

Limitations This chapter developed a modular neural framework for elastic simulation, in which the computation pipeline is decomposed into specialized and well-optimized neural modules. This design brings several advantages in scenario flexibility and the handling of physical constraints. At the same time, the current scope is limited to elastic simulation. Although Neural Modular Physics is intended as a general idea, extending it to other physical domains remains future work. In particular, fluid simulation would require a different modularization strategy aligned with the structure of traditional computational fluid dynamics solvers. Notably, fluid simulation is itself naturally modular, typically involving advection, force application, pressure projection, and boundary processing, which suggests a possible starting point for extending the framework beyond elasticity.

More broadly, the present results indicate that modularization can be a useful inductive bias for learned physics, but the appropriate module structure remains domain dependent. Identifying such structure in other classes of physical systems, and designing stable training strategies that preserve specialization across modules, remains an important direction for future work.

This chapter presented *Neural Modular Physics* (NMP), a modular framework that combines the physical reliability of classical simulators with the flexibility of neural simulators. With a specialized architecture and a two-stage training strategy, NMP transforms the computation pipeline of elastic simulation into a sequence of neural modules. This modular design exposes intermediate physical quantities, supports plug-and-play interchange between analytic and learned components, and enables analytic computation of physical constraints, capabilities that are absent in prior monolithic neural simulators. Experiments show that NMP achieves stronger accuracy and long-horizon stability than neural and hybrid baselines, while maintaining good generalization across initial conditions and resolutions and providing improved scenario flexibility and efficiency relative to traditional simulators.

Within the overall thesis, this chapter provides the data-adaptive counterpart to the differentiability-aware simulation frameworks developed earlier. If the previous chapters show how physical simulation can be made usable for optimization and co-design, this chapter shows how such frameworks can be augmented with learned components while preserving their underlying structure and interpretability.

Chapter 6

Conclusion and Future Work

The preceding chapters developed the main technical contributions of this thesis, showing how physical simulation can be reformulated to support optimization, co-design, and structured data adaptation. Together, these results suggest a broader view of simulation: not only as a tool for forward analysis, but as a computational foundation for reasoning and decision-making in physical systems. At the same time, they also point to several directions in which the present framework can be extended. This chapter first outlines promising future directions that build naturally on the ideas developed in the thesis, and then concludes by summarizing the central contributions and broader implications of this work.

6.1 Future Work

6.1.1 Physics Foundation Modules

A natural next step is the development of *physics foundation modules*: a small set of reusable, composable building blocks for physical simulation that can be shared across domains. The results of this thesis suggest that differentiability and modularity are most powerful when they are built into the simulator itself rather than added afterward. Building on this idea, physics foundation modules would serve as modular and composable physics priors, enabling components to be reused across different physical systems while remaining differentiable and scalable by design. At the same time, such modules could remain data-adaptive without sacrificing structure, allowing learning to correct missing or uncertain physics while preserving the invariants and constraints that keep simulation reliable. Under this view, machine learning provides parameterization and adaptation mechanisms, while physics modules provide structure and inductive bias.

6.1.2 Learning in the Real World

A second direction is learning in the real world under partial and noisy observation. In simulation, the full physical state is available: positions, velocities, internal forces, and contact events can all be accessed directly. Real systems, by contrast, are observed only indirectly through images, video, and sparse sensor signals such as force, inertial, or tactile measurements. Much of the relevant physical state is therefore hidden, often under occlusion

or sensor noise. An important research goal is to bridge this gap by inferring useful latent state from partial observations and coupling that inference with physics-grounded models for prediction and control. This direction suggests a close integration between simulation and computer vision, including techniques such as 3D reconstruction, tracking, correspondence, and generative video modeling, so that physical systems can be learned and controlled reliably under realistic sensing conditions.

6.1.3 Toward Intelligent Physical Systems

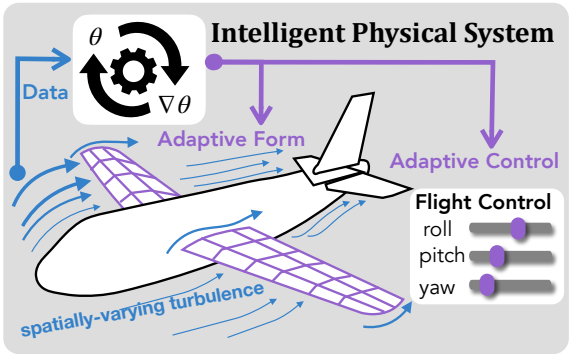


Figure 6.1: Conceptual illustration of an intelligent physical system in aerospace, where data are used to update the internal model and drive both adaptive form and adaptive control under changing flow conditions.

A third direction is the development of intelligent physical systems that jointly optimize form, control, and physical models, and continue to adapt after deployment. Rather than treating geometry, controller design, and model calibration as separate stages, such systems would co-design them within a shared computational framework and update them from data during operation. In this setting, a deployed system could detect when reality departs from prediction, update its internal model, and then revise how it acts. An aircraft provides a representative example: with onboard sensing, the system could detect when the true aerodynamic environment differs from its prediction, update its internal model, and then adjust both its control policy and the behavior of flexible wing structures to maintain efficiency and robustness. Figure 6.1 illustrates this idea schematically, showing a closed loop in which data drive adaptation of both form and control in response to spatially varying turbulence.

This principle has implications across many engineering domains. In mechanical systems, it suggests structures whose deformation directly encodes sensing information. In wearable and human-centered design, it suggests devices such as soft exoskeletons that adapt assistance to posture and motion. In aerospace, it suggests adaptive aeroelastic systems whose structural response and control policy jointly respond to changing conditions such as turbulence. Figure 6.2 illustrates these possibilities through three representative examples: intelligent metamaterial sensing, a soft assistive exoskeleton, and an adaptive aeroelastic wing.

More broadly, these directions point toward a future in which physical systems do not merely execute precomputed designs, but continuously reason, adapt, and improve through the integration of simulation, optimization, learning, and real-world data.

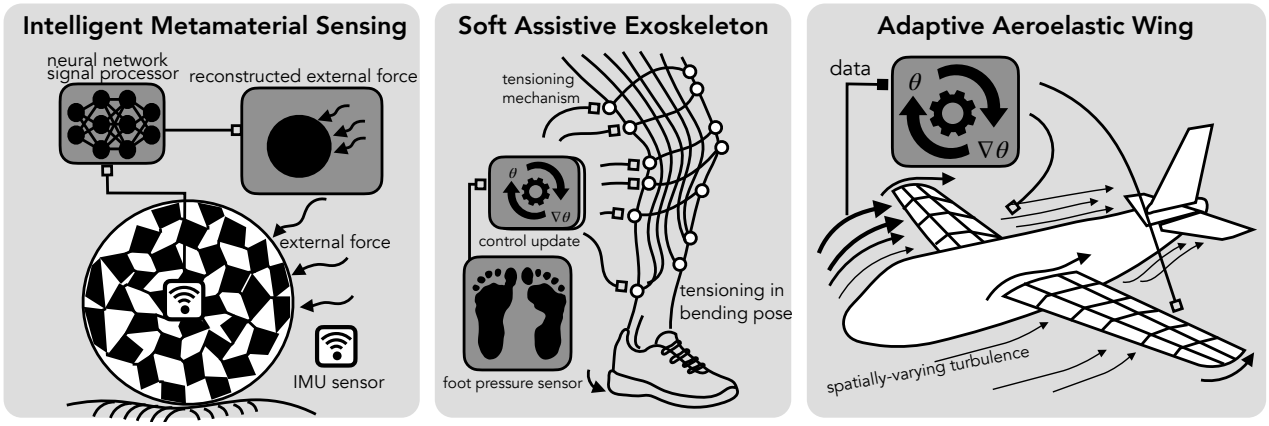


Figure 6.2: Representative directions for intelligent physical systems, including intelligent metamaterial sensing, soft assistive exoskeletons, and adaptive aeroelastic wings. In each case, sensing, physical structure, and control are integrated within a shared adaptive framework.

6.2 Conclusion

This thesis developed computational foundations for Physical AI by studying how simulation, optimization, and learning can be unified for intelligent physical systems. The central motivation was that, as simulation moves beyond visual computing into domains such as engineering, robotics, and biomedical design, the criterion for success shifts from visual plausibility to physical consequence. In such settings, high-fidelity simulators remain indispensable because they provide physically grounded and controllable models, but they are often limited to forward prediction and are not naturally suited to optimization, control, or adaptation from data. Conversely, learning-based methods offer differentiability and data adaptability, but often lack the physical structure and reliability required for real-world deployment.

To address this gap, the thesis developed three connected ideas. First, it showed that physical models must often be reformulated in a differentiability-aware manner so that they remain both accurate and usable for optimization. This idea was developed in the context of fluid-solid systems, where interface behavior, topology, and boundary conditions must be represented in a way that supports both physical fidelity and gradient-based design. Second, the thesis demonstrated how differentiable simulation can serve as a computational substrate for optimization and co-design in dynamic fluidic systems, enabling joint reasoning over geometry and control. Third, it introduced a modular perspective on data adaptation, showing how learned components can augment physical solvers while preserving their structure, interpretability, and computational interfaces.

Taken together, these contributions support the broader claim of the dissertation: that progress toward Physical AI requires a synthesis of physical accuracy, differentiability, and data adaptability within a common computational framework. More broadly, this thesis argues that simulation should no longer be viewed only as a tool for forward analysis, but as a foundation for reasoning, design, and adaptation in physical systems.

Bibliography

[1] Yifei Li, Tao Du, Sangeetha Grama Srinivasan, Kui Wu, Bo Zhu, Eftychios Sifakis, and Wojciech Matusik. Fluidic topology optimization with an anisotropic mixture model. *ACM Trans. Graph.*, nov 2022. ISSN 0730-0301. doi:10.1145/3550454.3555429. URL https://doi.org/10.1145/3550454.3555429.

[2] Yifei Li, Yuchen Sun, Pingchuan Ma, Eftychios Sifakis, Tao Du, Bo Zhu, and Wojciech Matusik. Neuralfluid: Neural fluidic system design and control with differentiable simulation, 2024.

[3] Yifei Li, Haixu Wu, Zeyi Xu, Tuur Stuyck, and Wojciech Matusik. Neural modular physics for elastic simulation, 2025. URL https://arxiv.org/abs/2512.15083.

[4] Krister Svanberg. The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2):359–373, 1987.

[5] Tao Du, Kui Wu, Andrew Spielberg, Wojciech Matusik, Bo Zhu, and Eftychios Sifakis. Functional optimization of fluidic devices with differentiable stokes flow. *ACM Trans. Graph.*, 39(6), December 2020. ISSN 0730-0301. doi:10.1145/3414685.3417795. URL https://doi.org/10.1145/3414685.3417795.

[6] Thomas Borrvall and Joakim Petersson. Topology optimization of fluids in stokes flow. *International journal for numerical methods in fluids*, 41(1):77–107, 2003.

[7] Alexander Clegg, Zackory Erickson, Patrick Grady, Greg Turk, Charles C. Kemp, and C. Karen Liu. Learning to collaborate from simulation for robot-assisted dressing. *IEEE Robotics and Automation Letters*, 5(2):2746–2753, 2020.

[8] Antony Jameson. Optimum aerodynamic design using cfd and control theory. *CFD Review*, 3, 06 1995. doi:10.2514/6.1995-1729.

[9] K Maute and M Allen. Conceptual design of aeroelastic structures by topology optimization. *Structural and Multidisciplinary Optimization*, 27(1-2):27–42, 2004.

[10] Pavel Ŝolín. *Partial differential equations and the finite element method*. John Wiley & Sons, 2005.

[11] Katja Wolff, Philipp Herholz, Verena Ziegler, Frauke Link, Nico Brügel, and Olga Sorkine-Hornung. Designing personalized garments with body movement. *Computer Graphics Forum*, 42(1):180–194, 2023. doi:https://doi.org/10.1111/cgf.14728. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14728.

[12] Donglai Xiang, Timur Bagautdinov, Tuur Stuyck, Fabian Prada, Javier Romero, Weipeng Xu, Shunsuke Saito, Jingfan Guo, Breannan Smith, Takaaki Shiratori, et al. Dressing avatars: Deep photorealistic appearance for physically simulated clothing. *ACM Transactions on Graphics (TOG)*, 41(6):1–15, 2022.

[13] Joe Alexandersen and Casper Schousboe Andreasen. A review of topology optimisation for fluid-based problems. *Fluids*, 5(1), 2020. ISSN 2311-5521. doi:10.3390/fluids5010029. URL https://www.mdpi.com/2311-5521/5/1/29.

[14] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. Add: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[15] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, July 2019. ISSN 1091-6490. doi:10.1073/pnas.1814058116.

[16] Yuan Yin, Vincent Le Guen, Jérémie Dona, Emmanuel de Bézenac, Ibrahim Ayed, Nicolas Thome, and Patrick Gallinari. Augmenting physical models with deep networks for complex dynamics forecasting. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124012, December 2021. ISSN 1742-5468. doi:10.1088/1742-5468/ac3ae5.

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

[18] Haixu Wu, Huakun Luo, Yuezhou Ma, Jianmin Wang, and Mingsheng Long. Ropinn: Region optimized physics-informed neural networks. *Advances in Neural Information Processing Systems*, 37:110494–110532, 2024.

[19] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[20] Zong-Yi Li, Daniel Z. Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *JMLR*, 2023.

[21] Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In *International Conference on Machine Learning*, pages 53681–53705. PMLR, 2024.

[22] Pingchuan Ma, Peter Yichen Chen, Bolei Deng, Joshua B Tenenbaum, Tao Du, Chuang Gan, and Wojciech Matusik. Learning neural constitutive laws from motion observations for generalizable pde dynamics. In *International Conference on Machine Learning*, pages 23279–23300. PMLR, 2023.

[23] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92 (3):88, 2022.

[24] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[25] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Mitigating propagation failures in physics-informed neural networks using retain-resample-release (r3) sampling. In *International Conference on Machine Learning*, 2023.

[26] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 2024.

[27] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH Comput. Graph.*, 21(4):205–214, August 1987. ISSN 0097-8930.

[28] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, 1998.

[29] Rahul Narain, Armin Samii, and James F O'brien. Adaptive anisotropic remeshing for cloth simulation. *ACM transactions on graphics (TOG)*, 31(6):1–10, 2012.

[30] Shiwei Zhou and Qing Li. A variational level set method for the topology optimization of steady-state navier-stokes flow. *Journal of Computational Physics*, 227(24):10178–10195, 2008.

[31] Yongbo Deng, Zhenyu Liu, and Yihui wu. Topology optimization of steady and unsteady incompressible navier–stokes flows driven by body forces. *Structural and Multidisciplinary Optimization*, 47, 11 2012. doi:10.1007/s00158-012-0847-8.

[32] Gil Ho Yoon. Topology optimization for stationary fluid-structure interaction problems using a new monolithic formulation. *International Journal for Numerical Methods in Engineering*, 82(5):591–616, 2010.

[33] Walter Jesus Paucar Casas and Renato Pavanello. Optimization of fluid-structure systems by eigenvalues gap separation with sensitivity analysis. *Applied Mathematical Modelling*, 42:269–289, 2017.

[34] Casper Schousboe Andreasen and Ole Sigmund. Topology optimization of fluid–structure-interaction problems in poroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 258:55–62, 2013.

[35] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable programming for physical simulation. In *ICLR*, 2020.

[36] Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *Advances in Neural Information Processing Systems (NeurIPS) Workshop*, 2022.

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

[38] Björn List, Li-Wei Chen, and Nils Thuerey. Learned turbulence modelling with differentiable fluid solvers: physics-based loss functions and optimisation horizons. *Journal of Fluid Mechanics*, 949, September 2022. ISSN 1469-7645. doi:10.1017/jfm.2022.738. URL http://dx.doi.org/10.1017/jfm.2022.738.

[39] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics. *ACM Trans. Graph.*, 41 (2), nov 2021. ISSN 0730-0301. doi:10.1145/3490168. URL https://doi.org/10.1145/3490168.

[40] Yiling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Differentiable simulation of soft multi-body systems. *Advances in Neural Information Processing Systems*, 34, 2021.

[41] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Efficient differentiable simulation of articulated bodies. In *International Conference on Machine Learning*, pages 8661–8671. PMLR, 2021.

[42] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *ICML*, 2020.

[43] Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C Lin. Differentiable fluids with solid coupling for learning and control. In *AAAI*, 2021.

[44] Zhehao Li, Qingyu Xu, Xiaohan Ye, Bo Ren, and Ligang Liu. Difffr: Differentiable sph-based fluid-rigid coupling for rigid body control. *ACM Trans. Graph.*, 42(6), Dec 2023. URL https://doi.org/10.1145/3618318.

[45] James K Guest and Jean H Prévost. Topology optimization of creeping fluid flows using a darcy–stokes finite element. *International Journal for Numerical Methods in Engineering*, 66(3):461–484, 2006.

[46] Niels Aage, Thomas H Poulsen, Allan Gersborg-Hansen, and Ole Sigmund. Topology optimization of large scale stokes flow problems. *Structural and Multidisciplinary Optimization*, 35(2):175–180, 2008.

[47] Vivien J. Challis and James K. Guest. Level set topology optimization of fluids in stokes flow. *International Journal for Numerical Methods in Engineering*, 79(10):1284–1308, 2009.

[48] Tadayoshi Matsumori, Tsuguo Kondoh, Atsushi Kawamoto, and Tsuyoshi Nomura. Topology optimization for fluid–thermal interaction problems under constant input power. *Structural and Multidisciplinary Optimization*, 47(4):571–581, 2013.

[49] Kentaro Yaji, Takayuki Yamada, Seiji Kubo, Kazuhiro Izui, and Shinji Nishiwaki. A topology optimization method for a coupled thermal–fluid problem using level set boundary expressions. *International Journal of Heat and Mass Transfer*, 81:878–888, 2015.

[50] Casper Schousboe Andreasen, Allan Roulund Gersborg, and Ole Sigmund. Topology optimization of microfluidic mixers. *International Journal for Numerical Methods in Fluids*, 61(5):498–513, 2009.

[51] George IN Rozvany. A critical review of established methods of structural topology optimization. *Structural and multidisciplinary optimization*, 37(3):217–237, 2009.

[52] Ole Sigmund and Kurt Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055, 2013.

[53] Joshua D Deaton and Ramana V Grandhi. A survey of structural and multidisciplinary continuum topology optimization: post 2000. *Structural and Multidisciplinary Optimization*, 49(1):1–38, 2014.

[54] Haixiang Liu, Yuanming Hu, Bo Zhu, Wojciech Matusik, and Eftychios Sifakis. Narrow-band topology optimization on a sparsely populated grid. *ACM Trans. Graph.*, 37(6), dec 2018. ISSN 0730-0301.

[55] Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. Designing inflatable structures. *ACM Transactions on Graphics (TOG)*, 33(4):63, 2014.

[56] Jun Wu, Niels Aage, Rüdiger Westermann, and Ole Sigmund. Infill optimization for additive manufacturing—approaching bone-like porous structures. *IEEE Transactions on Visualization and Computer Graphics*, 24(2):1127–1140, 2018.

[57] Bo Zhu, Mélina Skouras, Desai Chen, and Wojciech Matusik. Two-scale topology optimization with microstructures. *ACM Trans. Graph.*, 36(4), jul 2017. ISSN 0730-0301.

[58] Julian Panetta, Qingnan Zhou, Luigi Malomo, Nico Pietroni, Paolo Cignoni, and Denis Zorin. Elastic textures for additive fabrication. *ACM Trans. Graph.*, 34(4):135:1–135:12, July 2015. ISSN 0730-0301.

[59] Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. Microstructures to control elasticity in 3d printing. *ACM Trans. Graph.*, 34(4), jul 2015. ISSN 0730-0301.

[60] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.

[61] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pages 3424–3433. PMLR, 2017.

[62] Yaser Afshar, Saakaar Bhatnagar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64:525 – 545, 2019.

[63] Jinxian Liu, Ye Chen, Bingbing Ni, Wei Ren, Zhenbo Yu, and Xiaoyang Huang. Fast fluid simulation via dynamic multi-scale gridding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 1675–1682, 2023.

[64] 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.

[65] 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.

[66] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International conference on machine learning*, pages 4470–4479. PMLR, 2018.

[67] Oshri Halimi, Egor Larionov, Zohar Barzelay, Philipp Herholz, and Tuur Stuyck. Physgraph: Physics-based integration using graph neural networks. *arXiv preprint arXiv:2301.11841*, 2023.

[68] 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.

[69] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3:218 – 229, 2019.

[70] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[71] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.

[72] Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. *Advances in neural information processing systems*, 31, 2018.

[73] Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. Is a modular architecture enough? *Advances in Neural Information Processing Systems*, 35:28747–28760, 2022.

[74] Devon Jarvis, Richard Klein, Benjamin Rosman, and Andrew M Saxe. On the specialization of neural modules. In *International Conference on Learning Representations*, 2023.

[75] Cyril W Hirt and Billy D Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of computational physics*, 39(1):201–225, 1981.

[76] Stanley Osher Ronald Fedkiw and Stanley Osher. Level set methods and dynamic implicit surfaces. *Surfaces*, 44(77):685, 2002.

[77] Mark Sussman and Elbridge Gerry Puckett. A coupled level set and volume-of-fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of computational physics*, 162(2):301–337, 2000.

[78] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational physics*, 183(1):83–116, 2002.

[79] Yijing Li and Jernej Barbič. Stable anisotropic materials. *IEEE Transactions on Visualization and Computer Graphics*, 21(10):1129–1137, 2015.

[80] P.J. Basser, J. Mattiello, and D. Lebihan. Estimation of the effective self-diffusion tensor from the nmr spin echo. *Journal of Magnetic Resonance, Series B*, 103(3): 247–254, 1994. ISSN 1064-1866.

[81] Taylor Patterson, Nathan Mitchell, and Eftychios Sifakis. Simulation of complex nonlinear elastic bodies using lattice deformers. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.

[82] Christie Alappat, Achim Basermann, Alan R. Bishop, Holger Fehske, Georg Hager, Olaf Schenk, Jonas Thies, and Gerhard Wellein. A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. *ACM Trans. Parallel Comput.*, 7(3), June 2020. ISSN 2329-4949. doi:10.1145/3399732. URL https://doi.org/10.1145/3399732.

[83] F. J. Gaspar, J. L. Gracia, F. J. Lisbona, and C. W. Oosterlee. Distributive smoothers in multigrid for problems with dominating grad–div operators. *Numerical Linear Algebra with Applications*, 15(8):661–683, 2008. doi:https://doi.org/10.1002/nla.587. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.587.

[84] Georgia D. Ntouni, Athanasios S. Lioumpas, and Konstantina S. Nikita. Reliable and energy-efficient communications for wireless biomedical implant systems. *IEEE Journal of Biomedical and Health Informatics*, 18(6):1848–1856, 2014.

[85] David Erickson and Dongqing Li. Integrated microfluidic devices. *Analytica Chimica Acta*, 507(1):11–26, 2004.

[86] Kenzo Nonami, Farid Kendoul, Satoshi Suzuki, Wei Wang, and Daisuke Nakazawa. *Autonomous flying robots: unmanned aerial vehicles and micro aerial vehicles*. Springer Science & Business Media, 2010.

[87] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[88] Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. *ICLR*, 2023.

[89] Pingchuan Ma, Tao Du, John Z Zhang, Kui Wu, Andrew Spielberg, Robert K Katzschmann, and Wojciech Matusik. Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *ACM Transactions on Graphics (TOG)*, 40(4):132, 2021.

[90] Jos Stam. Stable fluids. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, page 121–128, 1999.

[91] Robert Bridson. *Fluid simulation for computer graphics*. AK Peters/CRC Press, Boca Raton, FL, USA, 2 edition, 2015.

[92] Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8:2182–2189, 1965.

[93] Yen Ting Ng, Chohong Min, and Frédéric Gibou. An efficient fluid–solid coupling algorithm for single-phase flows. *Journal of Computational Physics*, 228(23):8807–8829, 2009.

[94] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 — seamless operability between c++11 and python, 2016. https://github.com/pybind/pybind11.

[95] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.

[96] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[97] Nikolaus Hansen and Stefan Kern. Evaluating the cma evolution strategy on multimodal test functions. volume 3242, pages 282–291, 2004.

[98] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid poisson solver for fluids simulation on large grids. SCA '10, page 65–74, 2010.

[99] Pascal Friederich, Florian Häse, Jonny Proppe, and Alán Aspuru-Guzik. Machine-learned potentials for next-generation matter simulations. *Nature Materials*, 20(6):750–761, 2021. doi:10.1038/s41563-020-0777-6. Review Article, published online 27 May 2021.

[100] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2018.10.045.

[101] Yanan Guo, Xiaoqun Cao, Bainian Liu, and Mei Gao. Solving partial differential equations using deep learning and physical constraints. *Applied Sciences*, 10(17), 2020. ISSN 2076-3417. doi:10.3390/app10175917.

[102] Yifan Wang and Linlin Zhong. Nas-pinn: neural architecture search-guided physics-informed neural network for solving pdes. *Journal of Computational Physics*, 496: 112603, 2024.

[103] Tuur Stuyck, Gene Wei-Chin Lin, Egor Larionov, Hsiao-yu Chen, Aljaz Bozic, Nikolaos Sarafianos, and Doug Roble. Quaffure: Real-time quasi-static neural hair simulation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 239–249, 2025.

[104] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[105] Claudio Gentile and Manfred KK Warmuth. Linear hinge loss and average margin. *Advances in neural information processing systems*, 11, 1998.

[106] Keenan Crane, Ulrich Pinkall, and Peter Schröder. Robust fairing via conformal curvature flow. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

[107] Bob lifebuoy duck. https://www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/. Accessed: 2025-01-28.

[108] Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics (TOG)*, 37(2):1–15, 2018.

[109] Minkai Xu, Jiaqi Han, Aaron Lou, Jean Kossaifi, Arvind Ramanathan, Kamyar Azizzadenesheli, Jure Leskovec, Stefano Ermon, and Anima Anandkumar. Equivariant graph neural operator for modeling 3D dynamics. In *International Conference on Machine Learning*, pages 55015–55032. PMLR, 2024.