Building World Models with Neural Physics

by

Pingchuan Ma

B.Eng., Nankai University (2019)
S.M., Massachusetts Institute of Technology (2023)

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

February 2025

Authored by: Pingchuan Ma
Department of Electrical Engineering and Computer Science
January 20, 2025

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

Building World Models with Neural Physics

by

Pingchuan Ma

Submitted to the Department of Electrical Engineering and Computer Science
on January 20, 2025, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

World models learn the dynamics of environments in a data-driven manner, enhancing performance and efficiency in downstream tasks such as control, design, recognition, and generation, thanks to cost-effective simulation and differentiability. A pre-trained world model should ideally (1) accurately simulate ground-truth dynamics, (2) adapt easily to novel configurations, and (3) generalize across diverse physical effects. Previous attempts in this area have either utilized differentiable model-based physics with few parameters exposed or trained for specific scenarios with minimal physical priors integrated. These world models fall short of their objectives, limiting their applicability in real-world accurate-critic deployments and scalability to larger pre-trained world models. In this thesis, we aim to build world models with *neural physics*, a hybrid neural-physics framework that models the basic dynamics with differentiable physics while learning all additional modules through neural networks. By integrating neural physics, the world models adhere closely to physical principles while efficiently learning diverse effects. The modular structure of neural physics allows world models to generalize to novel configurations simply by installing different pre-trained neural modules. We will demonstrate the effectiveness of this novel framework in applications such as reconstruction, robotic control, and scientific discovery.

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

Previously Published Materials

Chapter 2 revises a previous publication [113]: Pingchuan Ma, Tao Du, Joshua B. Tenenbaum, Wojciech Matusik, and Chuang Gan. "RISP: Rendering-Invariant State Predictor with Differentiable Simulation and Rendering for Cross-Domain Parameter Estimation." In International Conference on Learning Representations. 2021.

Chapter 3 revises a previous publication [115]: 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. 2023.

Chapter 4 revises a previous manuscript [116]: Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. "LLM and Simulation as Bilevel Optimizers: A New Paradigm to Advance Physical Scientific Discovery." In International Conference on Machine Learning. 2024.

Acknowledgments

Pursuing a Ph.D. is both a challenging and rewarding journey, and I am fortunate to have supportive company throughout this path.

First and foremost, I would like to express my sincere gratitude to Prof. Wojciech Matusik, my thesis supervisor and research advisor. Wojciech consistently encourages me to view research as more than just a project but as an avenue to open new opportunities, and this perspective has profoundly influenced my research taste. We have spent countless hours discussing and even debating the future of research almost daily, during which I have benefited from his wisdom in calibrating my research direction and making significant career decisions. He has also trained me to defend my research ideas confidently against criticism, enhancing my courage during presentations and confrontations. The Ph.D. journey is a relatively short period in one's life, and I am incredibly fortunate to have had the opportunity to distill knowledge from Wojciech.

I would also like to thank Prof. Daniela Rus and Prof. Kaiming He for serving on my thesis committee. My collaboration with Daniela traces back to my first publication at MIT. Her extensive knowledge and passion for robotics helped me "keep my gradient" throughout my Ph.D. journey. Kaiming has been a perceptive mentor who guided me in elevating this thesis to a higher level. He imparted a deep understanding and vision of learning and physics, which expanded my thinking beyond the scope of this thesis. I sincerely hope for more opportunities to collaborate in the future.

I am also grateful to my mentors who worked alongside me throughout my Ph.D. journey. Prof. Tao Du, a senior labmate, guided me during my first few projects. He is highly hands-on and imparted invaluable lessons on becoming a proficient researcher, from identifying and pitching a compelling problem to finding elegant solutions. Beyond academia, he has been a good friend, offering insights and assisting me in making informed decisions in life. Dr. Jie Xu, another senior labmate, also served as my mentor at the NVIDIA Seattle Robotics Lab. He worked closely with me on my internship project, providing guidance and a comprehensive vision of the entire pipeline of robotic research. Prof. Chuang Gan was my mentor at the MIT-IBM Watson AI Lab during my first internship after joining MIT. Our

collaboration began there and has continued to flourish to this day, contributing significantly to my growth and experience in the field. I also would like to express my gratitude to my undergraduate mentors: Prof. Bo Ren, Dr. Zherong Pan, Prof. Dinesh Manocha, Yao Zhou, and Prof. Ping Luo. Their patience and tolerance towards me as a junior researcher have been instrumental in my journey to gaining admission to MIT.

I would like to thank all my other collaborators: Stan Birchfield, Peter Yichen Chen, Bolei Deng, Yilun Du, Dieter Fox, Yuchao Gu, Minghao Guo, Ankur Handa, Changyu Hu, Josie Hughes, Joshua Jacob, Robert Katzschmann, Byungchul Kim, Yichen Li, Yifei Li, Chunru Lin, Ziming Liu, Mike Michelis, Yashraj Narang, Elvis Nava, Crystal Owens, Juan Salazar, Eftychios Sifakis, Andrew Spielberg, Shinjiro Sueda, Yuchen Sun, Zhiqing Sun, Max Tegmark, Joshua B. Tenenbaum, Yunsheng Tian, Sebastien Wah, Bohan Wang, Tsun-Hsuan Wang, Yixuan Wang, Bowen Wen, Kui Wu, Zhou Xian, Xiaoyu Xiong, Wei Yang, John Zhang, Tianyuan Zhang, Wei Zhang, Yu Zhang, Allan Zhao, Juntian Zheng, and Bo Zhu. None of my research works would have been possible without the incredible support and contributions from them.

In addition to my research pursuits, I received invaluable love and support from my friends. I fondly remember the times spent enjoying barbecues and end-of-year holiday dinners with my labmates. I also cherished attending the CGGAR retreats and participating in the ABCDEFG ski events with the MIT Computer Graphics Group. Additionally, the summer activities with the NVIDIA Seattle Robotics Lab interns were memorable highlights. I enjoyed omakase, snowboarding, tennis, Texas hold 'em, and Mahjong with Ji Lin, Siyu Yao, Lijie Fan, Tianhong Li, and Wendi Gao. I also found great joy in playing Avalon the Resistance with Yonglong Tian, Lei Xu, Hanrui Wang, and Yue Wang. I shared many drinks with Lijie Chen, Qicheng Zhao, Toru Lin, Baian Chen, Brabeeba Wang, Zihao Xu, and Aoxiao Zhong, and explored tons of restaurants with Jiayuan Mao, Xinyi Gu, Minghao Guo, Tianyu Li, Xiaolin Fang, Ching-Yun Ko, Yifei Li, Tsun-Hsuan Wang, Shuang Li, and Heng Yang. I had amazing roommate-ship with Tiancheng Yu, Haotian Tang, and Yian Wang. Additionally, I have maintained long-term friendships in this foreign country with Yunsheng Tian, Ruihan Yang, and Yujun Shi, dating back to our undergraduate years. During my Ph.D. journey, I developed an interest in bodybuilding

and greatly appreciate the spotting and companionship of Tiancheng Yu, Guangxuan Xiao, Bohan Wang, and Bolei Deng. Also, I began exploring portrait photography, where I was fortunate to meet many patient models who supported me and contributed to my growth. I am especially grateful to Gege Qian, my first portrait model, for her encouragement and patience. It is often said that finding genuine friends in adulthood — those who truly celebrate your successes and empathize with your sorrows — is rare, and I am extremely lucky to have found such wonderful people in my life.

Finally, I want to express my most sincere gratitude to my family. I know that pursuing a Ph.D. abroad has been a great challenge for not only me but also my parents. However, they have supported every decision I have made without a moment's hesitation. Throughout my life, I have seized every opportunity to develop independence, but I have always known that it was only possible because of their unconditional love and support. At the end, I want to extend a special thank you to my late grandfather, Rujiao Ma. He passed away during the third year of my Ph.D. studies, and due to various complex issues, I was unable to attend his funeral. He invested greatly in my growth and always encouraged me to pursue higher education. I hope he would be proud of my achievements, and I wish him eternal peace.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Scientists have long been investigating effective methods to develop an efficient representation of the world, termed *world models* [51, 95]. World models allow agents to interact with their environments either for predicting future events [54, 57, 58, 80, 209, 211] or deriving gradients with respect to the current state or action [35, 37, 66, 67, 140, 142, 141, 191]. Typically, world models possess the capability to perceive multi-modal signals from the environment and respond to the input control signals with multi-modal feedback [51, 80, 113]. The availability of cost-effective future trajectories and derivatives significantly enhances data efficiency, robustness, and the performance of downstream tasks, including robotic control and design [57, 58, 113, 114], reasoning [51], and generation [192].

Recent works have developed physics-based world models using differentiable physical simulators or residual physics to model well-studied physical effects such as articulated rigid bodies [141, 199], deformable bodies [140, 142, 37, 191], fluid [35, 196], and multi-physics systems [101]. These world models generally align well with real-world physics by optimizing the pre-defined system parameters, allowing them to generalize effectively to unseen configurations. However, the reliance on manually designed physical engines and system parameters limits their generalizability to unmodeled physical effects. For instance, it requires significant effort, or may even be impossible, to simulate a fluidic environment using a differentiable deformable-body simulation built on dedicated finite

element methods. This limitation underscores the challenges in extending these models to diverse and complex physical phenomena.

On the other hand, another thread of research has been attempting to model the world with minimal physical priors using graph neural networks [154, 137] or latent dynamics models [53, 54]. These approaches aim to build underlying world representation without heavily relying on predefined physical parameters, instead leveraging the structure and learned representations to understand and predict dynamics. This can lead to more flexible models without the limitations of manually designed physics. However, due to the absence of fundamental physical principles, it may result in decreased sample efficiency, wasted computation, and compromised physical integrity.

Recent research has also explored the integration of physical priors into ordinary neural networks through the use of physics-informed neural networks (PINNs) [146, 144, 145, 81]. In this method, physical constraints are formulated as loss functions and are back-propagated to the neural network in a self-supervised manner. PINNs offer a novel perspective by combining both data and physical laws to model the physical world. However, their integration is often too weak to reliably support long-horizon downstream tasks. Additionally, PINNs are typically optimized for specific scene configurations within a particular time span, limiting their applicability to a broader range of scenarios.

1.2 Proposed Solution and Applications

In this thesis, we plan to rethink about the integration of neural networks and physics into world models and develop a hybrid *neural physics* method that enhances world models. Leveraging neural physics, we bridge world models and differentiable physics by refining foundational physical laws and learning additional complexities from data. Neural physics offers the following advantages:

1. **Accuracy.** Neural physics integrates fundamental physics into its architecture and parameterizes the most variable and critical components (e.g., material models or perceptual models) using neural networks, learning these parameters from data. Thus, neural physics benefits from the accuracy provided by both physical principles and

data-driven learning.

2. **Adaptability.** We design neural physics using a modular hierarchy, allowing users to freely select from a range of pre-trained neural physical modules and deploy them in a plug-and-play manner to adapt to novel configurations.

3. **Generalizability.** Neural physics disentangles various physical effects from fundamental physical laws, enabling it to generalize across a diverse range of physical phenomena.

In this thesis, we will explore multiple implementations and applications of neural physics, as detailed below:

1.2.1 Rendering-Invariant State Predictor (Chapter 2)

To begin with, we propose Rendering-Invariant State Predictor (RISP) [113], where we embed neural physics into the perceptual model in the world model. This work considers calibrating parameters characterizing a physical system's dynamic motion directly from a video whose rendering configurations are inaccessible. Existing solutions require massive training data or lack generalizability to unknown rendering configurations. We propose a novel approach that combines neural perceptual model and differentiable physics to address this problem. Our core idea is to train a rendering-invariant state-prediction (RISP) network that transforms image differences into state differences independent of rendering configurations, e.g., lighting, shadows, or material reflectance. To train this predictor, we formulate a new loss on rendering variances using gradients from differentiable rendering. Moreover, we present an efficient, second-order method to compute the gradients of this loss, allowing it to be integrated seamlessly into modern deep learning frameworks. We evaluate our method in multi-physics world models including rigid bodies and deformable bodies using 4 tasks: state estimation, system identification, imitation learning, and visuomotor control. We further demonstrate the efficacy of our approach on a real-world example: inferring the state and action sequences of a quadrotor from a video of its motion sequences. Compared with existing methods, our approach achieves significantly lower reconstruction errors and has better generalizability among unknown rendering configurations.

1.2.2 Neural Constitutive Laws (Chapter 3)

We propose a hybrid neural physics approach for learning generalizable world models from motion observations. Recent world models typically learn an end-to-end model that implicitly models both the governing partial differential equations (PDE) and constitutive models (or material models). Without explicit PDE knowledge, these models cannot guarantee physical correctness and have limited generalizability. We argue that the governing PDEs are often well-known and should be explicitly enforced rather than learned. Instead, constitutive models are particularly suitable for learning due to their data-fitting nature. To this end, we introduce a new neural physical framework termed neural constitutive laws (NCLaw) [115], which utilizes a network architecture that strictly guarantees standard constitutive priors, including rotation equivariance and undeformed state equilibrium. We connect NCLaw with a differentiable simulation and train the model by minimizing a loss function based on the difference between the simulation and the motion observation. We validate NCLaw on diverse large-deformation dynamical systems, ranging from solids to fluids. After training on *a single motion trajectory*, our method generalizes to new geometries, initial/boundary conditions, temporal ranges, and even multi-physics systems. On these extremely out-of-distribution generalization tasks, NCLaw is orders-of-magnitude more accurate than previous world models. Real-world experiments demonstrate our method's ability to learn constitutive laws from videos.

1.2.3 Scientific Generative Agent (Chapter 4)

Large Language Models (LLMs) have recently gained significant attention in scientific discovery for their extensive knowledge and advanced reasoning capabilities. However, they encounter challenges in effectively simulating observational feedback and grounding it with language to propel advancements in physical scientific discovery. Conversely, human scientists undertake scientific discovery by formulating hypotheses, conducting experiments, and revising theories through observational analysis. Inspired by this, we propose to enhance the knowledge-driven, abstract reasoning abilities of LLMs with the computational strength of world models. We introduce scientific generative agent (SGA) [116], a bilevel optimiza-

tion framework: LLMs act as knowledgeable and versatile thinkers, proposing scientific hypotheses and reason about discrete components, such as physics equations or molecule structures, and embed them inside a neural physical module. In the meantime, world models act as experimental platforms, providing observational feedback and optimizing the neural physical module via differentiability for continuous parts, such as physical parameters. We conduct extensive experiments to demonstrate our framework's efficacy in constitutive law discovery and molecular design, unveiling novel solutions that differ from conventional human expectations yet remain coherent upon analysis.

1.3 Thesis Contributions

This thesis and its core modules make the following key contributions:

- Rendering-Invariant State Predictor (Chapter 2) introduces a neural module cooperating with differentiable physics for narrowing down sim-to-real gap. The neural module is trained by minimizing the output variance with respect to different rendering parameters. With the efficient differentiable workflow, we train Render-Invariant State Predictor to close the sim-to-real gap with second-order optimization. We demonstrate the effectiveness of Rendering-Invariant State Predictor on 4 different environments ranging from complex articulated robotic hand to elastic rope.

- Neural Constitutive Laws (Chapter 3) introduces a set of neural networks modeling the mapping corresponding to the physical laws so that it parameterizes a variety of physical phenomena within a unified representation. Neural Constitutive Laws coordinates the rotational equivariance and static-state equilibrium by baking them into the neural architecture in order to improve the performance and stabilize the training. Neural Constitutive Laws provides the first unified neural parameterization, opening gates for more accurate sim-to-real calibration, faster physical simulation, and better understanding of the physical world.

- Scientific Generative Agent (Chapter 4) introduces an iterative workflow for scientific discovery with bilevel optimization combined by LLM and physical simulation. Sci-

entific Generative Agent interfaces between a natural language-based generalist and a code-based domain expert and cascades them for multiple scientific discovery problems. We proved that Scientific Generative Agent is more accurate, generalizable, and scalable than traditional symbolic regression by searching for novel constitutive laws. Additionally, the performance on molecular design tasks of Scientific Generative Agent shows competitive multi-discipline ability.

1.4 Thesis Impact

We believe that incorporating neural physics into world models may significantly enhance their performance, particularly in speed, efficiency, and physical realism. Traditional modeling of world models often either strictly enforces connections between different modules without any flexibility or opts for a completely end-to-end training approach, disregarding module connections entirely. We anticipate that neural physics will illuminate new possibilities for modularizing the construction pipeline of world models.

Secondly, we anticipate that neural physics will positively impact the controllability of world models. The real world encompasses a wide range of conditions, including physical interactions such as contact and intelligent interactions like conversation. Through these diverse and heterogeneous interactions, the world model creates a dynamic environment for agents to learn and evolve. By incorporating neural modules into physical simulation, the model can accept more versatile control signals, and by introducing modularization into world models, it can enforce physical constraints more strictly. We expect that the interoperability and interactability offered by neural physics will create new application and deployment scenarios for world models.

Lastly, this thesis highlights the potentially broad applications of world models in fields like robotics, scientific discovery, and general artificial intelligence. As a static repository of physical dynamics, the world model encompasses all resources necessary for the development and interaction of intelligence. Enhanced by neural physics, world models can achieve greater physical realism and efficiency, enabling more applications beyond mere reconstruction. Robotics, illustrated in this thesis, is a direct application

where an agent interacts with the model and plans actions accordingly. Accurate simulation of physical dynamics significantly reduces the costs associated with scientific discovery and accelerates iteration processes. General artificial intelligence seeks to model human-level intelligence and explore paths toward super-human capabilities. World models provide an ideal environment for studying the evolution and interaction of agents.

In short, this thesis proposes neural physical methods for building holistic world models and concretely illustrates potential new applications through examples. We hope this thesis serves as a foundation for the development of next-generation physical intelligence.

Chapter 2

RISP: Rendering-Invariant State Predictor with Differentiable Simulation and Rendering for Cross-Domain Parameter Estimation

2.1 Introduction

Reconstructing dynamic information about a physical system directly from a video has received considerable attention in the robotics, machine learning, computer vision, and graphics communities. This problem is fundamentally challenging because of its deep coupling among physics, geometry, and perception of a system. Traditional solutions like motion capture systems [180, 131, 143] can provide high-quality results but require prohibitively expensive external hardware platforms. More recent development in differentiable simulation and rendering provides an inexpensive and attractive alternative to the motion capture systems and has shown promising proof-of-concept results [126]. However, existing methods in this direction typically assume the videos come from a *known* renderer. Such an assumption limits their usefulness in inferring dynamic information from an *unknown* rendering domain, which is common in real-world applications due to the discrepancy be-

tween rendering and real-world videos. Existing techniques for aligning different rendering domains, e.g., CycleGAN [214], may help alleviate this issue, but they typically require access to the target domain with massive data, which is not always available. To our best knowledge, inferring dynamic parameters of a physical system directly from videos under *unknown* rendering conditions remains far from being solved, and our work aims to fill this gap.

We propose a novel approach combining three ideas to address this challenge: domain randomization, state estimation, and rendering gradients. Domain randomization is a classic technique for transferring knowledge between domains by generating massive samples whose variances can cover the discrepancy between domains. We upgrade it with two key innovations: First, we notice that image differences are sensitive to changes in rendering configurations, which shadows the rendering-invariant, dynamics-related parameters that we genuinely aim to infer. This observation motivates us to propose a *rendering-invariant state predictor* (RISP) that extracts state information of a physical system from videos. Our second innovation is to leverage *rendering gradients* from a differentiable renderer. Essentially, requiring the output of RISP to be agnostic to rendering configurations equals enforcing its gradients for rendering parameters to be zero. We propose a new loss function using rendering gradients and show an efficient method for integrating it into deep learning frameworks.

Putting all these ideas together, we develop a powerful pipeline that effectively infers parameters of a physical system directly from video input under random rendering configurations. We demonstrate the efficacy of our approach on a variety of challenging tasks evaluated in four environments (Sec. 2.4 and Fig. 2-1) as well as in a real-world application (Fig. 2-4). The experimental results show that our approach outperforms the state-of-the-art techniques by a large margin in most of these tasks due to the inclusion of rendering gradients in the training process.

In summary, our work makes the following contributions:

- We investigate and identify the bottleneck in inferring state, system, and control parameters of physical systems from videos under various rendering configurations (Sec. 2.3.1);

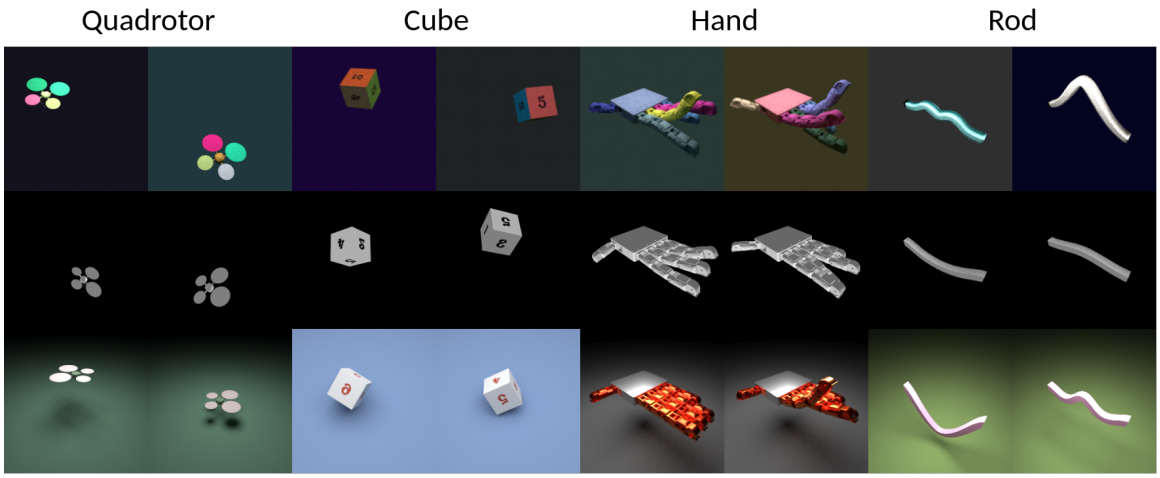


Figure 2-1: A gallery of our four environments (left to right) across three rendering domains (top to bottom). For each environment, we train a RISP with images under varying lighting, background, and materials generated from a differentiable render (top). Each environment then aims to find proper system and control parameters to simulate and render the physical system (middle) so that it matches the dynamic motion of a reference video (bottom) with unknown rendering configurations. We deliberately let three rows use renderers with vastly different rendering configurations.

- We propose a novel solution combining domain randomization, state estimation, and rendering gradients to achieve generalizability across rendering domains (Sec. 2.3.2);
- We demonstrate the efficacy of our approach on several challenging tasks in both simulation and real-world environments (Sec. 2.4).

2.2 Related Work

Differentiable simulation Differentiable simulation equips traditional simulation with gradient information. Such additional gradient information connects simulation tasks with numerical optimization techniques. Previous works have demonstrated the power of gradients from differentiable simulators in rigid-body dynamics [47, 32, 30, 199, 64, 141], deformable-body dynamics [37, 70, 36, 66, 45, 55, 114, 142], fluids [35, 121, 67], and co-dimensional objects [140, 104]. We make heavy use of differentiable simulators in this work but our contribution is orthogonal to them: we treat differentiable simulation as a black box, and our proposed approach is agnostic to the choice of simulators.

Differentiable rendering Differentiable rendering offers gradients for rendering inputs, e.g., lighting, materials, or shapes [149, 97, 73]. The state-of-the-art differentiable renderers [98, 127] are powerful in handling gradients even with global illumination or occlusion. Our work leverages these renderers but with a different focus on using their gradients as a physics prior in a learning pipeline.

Domain randomization The intuition behind domain randomization [171, 135, 4, 153, 168] is that a model can hopefully cross the domain discrepancy by seeing a large amount of random data in the source domain. This often leads to robust but conservative performance in the target domain. The generalizability of domain randomization comes from a more *robust* model that attempts to absorb domain discrepancies by behaving conservatively, while the generalizability of our method comes from a more *accurate* model that aims to match first-order gradient information.

2.3 Method

Given a video showing the dynamic motion of a physical system, our goal is to infer the unknown state, system, or control parameters directly from the video with partial knowledge about the physics model and rendering conditions. Specifically, we assume we know the governing equations of the physical system (e.g., Newton's law for rigid-body systems) and the camera position, but the exact system, control, or rendering parameters are not exposed.

To solve this problem, we propose a pipeline (Fig. 2-2) that consists of two components: 1) a differentiable simulation and rendering engine; 2) the RISP network. First, we use our engine to simulate and render the state of a physical system and outputs images under varying rendering configuration. Next, the RISP network learns to reconstruct the state information from these generated images. Putting these two components together, we have a pipeline that can faithfully recover dynamic information of a physical system from a new video with unseen rendering configurations.

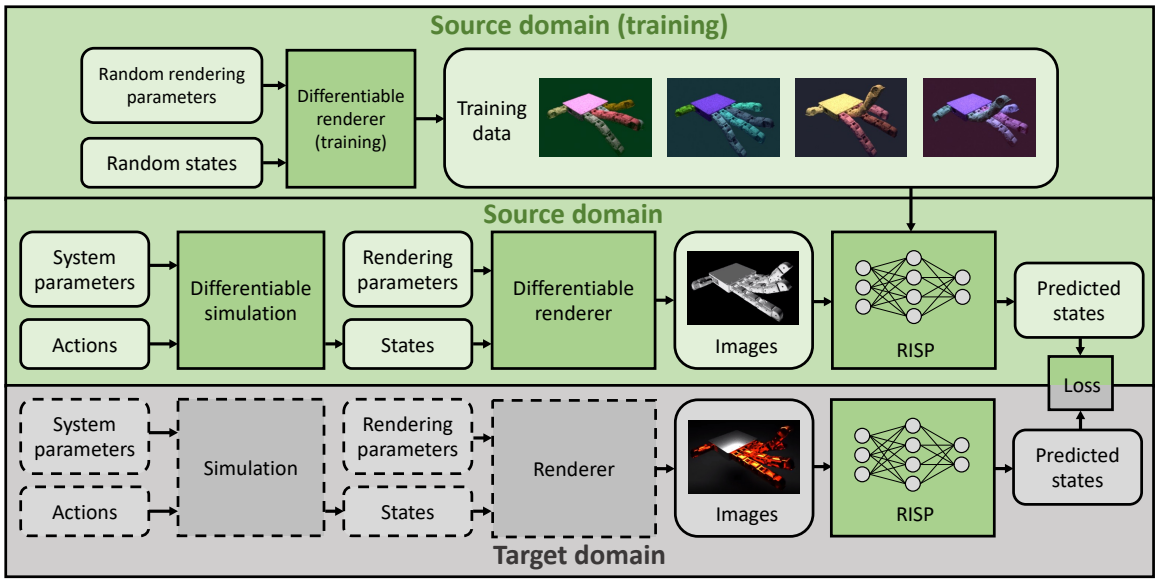


Figure 2-2: An overview of our method (Sec. 2.3). We first train RISP using images rendered with random states and rendering parameters (top). We then append RISP to the output of a differentiable renderer, leading to a fully differentiable pipeline from system and control parameters to states predicted from images (middle). Given reference images generated from unknown parameters (dashed gray boxes) in the target domain (bottom), we feed them to RISP and minimize the discrepancies between predicted states (rightmost green-gray box) to reconstruct the underlying system parameters, states, or actions.

2.3.1 Differentiable Simulation and Rendering Engine

Given a physical system with known dynamic model $\mathcal{M}$, we first use a differentiable simulator to simulate its states based on action inputs at each time step after time discretization:

$$\mathbf{s}_{i+1} = \mathcal{M}_{\boldsymbol{\phi}}(\mathbf{s}_i, \mathbf{a}_i), \quad \forall i = 0, 1, \cdots, N-1, \tag{2.1}$$

where N is the number of time steps in a rollout of physics simulation, and $\mathbf{s}_i$, $\mathbf{s}_{i+1}$ and $\mathbf{a}_i$ represent the state and action vectors at the corresponding time steps, respectively. The $\boldsymbol{\phi}$ vector encodes the system parameters in the model, e.g., mass, inertia, and elasticity. Next, we apply a differentiable renderer $\mathcal{R}$ to generate an image $\mathbf{I}_i$ for each state $\mathbf{s}_i$:

$$\mathbf{I}_i = \mathcal{R}_{\boldsymbol{\psi}}(\mathbf{s}_i), \quad \forall i = 0, 1, \cdots, N. \tag{2.2}$$

Here, $\boldsymbol{\psi}$ is a vector encoding rendering parameters whose gradients are available in the renderer $\mathcal{R}$. Examples of $\boldsymbol{\psi}$ include light intensity, material reflectence, or background color. By abuse of notation, we re-write the workflow of our simulation and rendering engine to a compact form:

$$\{\mathbf{I}_i\} = \mathcal{R}_{\boldsymbol{\psi}}[\underbrace{\mathcal{M}_{\boldsymbol{\phi}}(\mathbf{s}_0, \{\mathbf{a}_i\})}_{\{\mathbf{s}_i\}}]. \tag{2.3}$$

In other words, given an initial state $\mathbf{s}_0$ and a sequence of actions $\{\mathbf{a}_i\}$, we generate a sequence of states $\{\mathbf{s}_i\}$ from simulation and renders the corresponding image sequence $\{\mathbf{I}_i\}$. The task of recovering unknown information from a reference video $\{\mathbf{I}_i^{\text{ref}}\}$ can be formulated as follows:

$$\min_{\mathbf{s}_0, \{\mathbf{a}_i\}, \boldsymbol{\phi}, \boldsymbol{\psi}} \quad \mathcal{L}(\{\mathbf{I}_i^{\text{ref}}\}, \{\mathbf{I}_i\}), \tag{2.4}$$

$$\text{s.t.} \quad \{\mathbf{I}_i\} = \mathcal{R}_{\boldsymbol{\psi}}[\mathcal{M}_{\boldsymbol{\phi}}(\mathbf{s}_0, \{\mathbf{a}_i\})], \tag{2.5}$$

where $\mathcal{L}$ is a loss function penalizing the difference between the generated images and their references. Assuming that the simulator $\mathcal{M}$ and the renderer $\mathcal{R}$ are differentiable with respect to their inputs, we can run gradient-based optimization algorithms to solve Eqn. (2.4). This is essentially the idea proposed in ∇Sim, the state-of-the-art method for identifying parameters directly from video inputs [126]. Specifically, ∇Sim defines $\mathcal{L}$ as a norm on pixelwise differences.

One major limitation in Eqn. (2.4) is that it expects reasonably similar initial images $\{\mathbf{I}_i\}$ and references $\{\mathbf{I}_i^{\text{ref}}\}$ to successfully solve the optimization problem. Indeed, since the optimization problem is highly nonlinear due to its coupling between simulation and rendering, local optimization techniques like gradient-descent can be trapped into local minima easily if $\{\mathbf{I}_i\}$ and $\{\mathbf{I}_i^{\text{ref}}\}$ are not close enough. While ∇Sim has reported promising results when $\{\mathbf{I}_i\}$ and $\{\mathbf{I}_i^{\text{ref}}\}$ are rendered with moderately different $\boldsymbol{\psi}$, we found in our experiments that directly optimizing $\mathcal{L}$ defined on the image space rarely works when the two rendering domains are vastly different (Fig. 2-1). Therefore, we believe it requires a fundamentally different solution, motivating us to propose RISP in our method.

2.3.2 The RISP Network

The difficulty of generalizing Eqn. (2.4) across different rendering domains is partially explained by the fact that the loss $\mathcal{L}$ is defined on the differences in the image space, which is sensitive to changes in rendering configurations. To address this issue, we notice from many differentiable simulation papers that a loss function in the state space is fairly robust to random initialization [35, 104], inspiring us to redefine $\mathcal{L}$ in a state-like space. More concretely, we introduce the RISP network $\mathcal{N}$ that takes as input an image $\mathbf{I}$ and outputs a state prediction $\hat{\mathbf{s}} = \mathcal{N}(\mathbf{I})$. We then redefine the optimization problem in Eqn. (2.4) as follows (Fig. 2-2):

$$\min_{\mathbf{s}_0, \{\mathbf{a}_i\}, \boldsymbol{\phi}, \boldsymbol{\psi}} \quad \mathcal{L}(\mathcal{N}_{\boldsymbol{\theta}}(\{\mathbf{I}_i^{\text{ref}}\}), \mathcal{N}_{\boldsymbol{\theta}}(\{\mathbf{I}_i\})), \tag{2.6}$$

$$\text{s.t.} \quad \{\mathbf{I}_i\} = \mathcal{R}_{\boldsymbol{\psi}}[\mathcal{M}_{\boldsymbol{\phi}}(\mathbf{s}_0, \{\mathbf{a}_i\})]. \tag{2.7}$$

Note that the network $\mathcal{N}_{\boldsymbol{\theta}}$, parametrized by $\boldsymbol{\theta}$, is pre-trained and fixed in this optimization problem. Essentially, Eqn. (2.6) maps the two image sequences to the predicted state space, after which the standard gradient-descent optimization follows. A well-trained network $\mathcal{N}$ can be interpreted as an "inverse renderer" $\mathcal{R}^{-1}$ that recovers the rendering-invariant state vector regardless of the choice of rendering parameters $\boldsymbol{\psi}$, allowing Eqn. (2.6) to match the information behind two image sequences $\{\mathbf{I}_i\}$ and $\{\mathbf{I}_i^{\text{ref}}\}$ even when they are generated from different renderers $\mathcal{R}_{\boldsymbol{\psi}}$. Below, we present two ideas to train the network $\mathcal{N}$:

The first idea: domain randomization Our first idea is to massively sample state-rendering pairs $(\mathbf{s}_j, \boldsymbol{\psi}_j)$ and render the corresponding image $\mathbf{I}_j = \mathcal{R}_{\boldsymbol{\psi}_j}(\mathbf{s}_j)$, giving us a training set $\mathcal{D} = \{(\mathbf{s}_j, \boldsymbol{\psi}_j, \mathbf{I}_j)\}$. We then train $\mathcal{N}$ to minimize the prediction error:

$$\mathcal{L}^{\text{error}}(\boldsymbol{\theta}, \mathcal{D}) = \sum_{(\mathbf{s}_j, \boldsymbol{\psi}_j, \mathbf{I}_j) \in \mathcal{D}} \underbrace{\|\mathbf{s}_j - \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I}_j)\|_1}_{\mathcal{L}_j^{\text{error}}}. \tag{2.8}$$

The intuition is straightforward: $\mathcal{N}_{\boldsymbol{\theta}}$ learns to generalize over rendering configurations because it sees images generated with various rendering parameters $\boldsymbol{\psi}$. This is exactly the domain randomization idea [171], which we borrow to solve our problem across different

rendering domains.

The second idea: rendering gradients One major bottleneck in domain randomization is its needs for massive training data that spans the whole distribution of rendering parameters $\boldsymbol{\psi}$. Noting that a perfectly rendering-invariant $\mathcal{N}$ must satisfy the following condition:

$$\frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathcal{R}_{\boldsymbol{\psi}}(\mathbf{s}))}{\partial \boldsymbol{\psi}} \equiv \mathbf{0}, \quad \forall \mathbf{s}, \boldsymbol{\psi}, \tag{2.9}$$

we consider adding a regularizer to the training loss:

$$\mathcal{L}^{\text{train}}(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{L}^{\text{error}} + \gamma \underbrace{\sum_{(\mathbf{s}_j, \boldsymbol{\psi}_j, \mathbf{I}_j) \in \mathcal{D}} \| \frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathcal{R}_{\boldsymbol{\psi}_j}(\mathbf{s}_j))}{\partial \boldsymbol{\psi}_j} \|_{\text{F}}}_{\mathcal{L}^{\text{reg}}}, \tag{2.10}$$

where $\| \cdot \|_{\text{F}}$ indicates the Frobenius norm and γ a regularization weight. The intuition is that by suppressing this Jacobian to zero, we encourage the network $\mathcal{N}$ to flatten out its landscape along the dimension of rendering parameters $\boldsymbol{\psi}$, and invariance across rendering configurations follows. To implement this loss, we apply the chain rule:

$$\frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathcal{R}_{\boldsymbol{\psi}_j}(\mathbf{s}_j))}{\partial \boldsymbol{\psi}_j} = \frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I}_j)}{\partial \boldsymbol{\psi}_j} = \frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I}_j)}{\partial \mathbf{I}_j} \frac{\partial \mathbf{I}_j}{\partial \boldsymbol{\psi}_j}, \tag{2.11}$$

where the first term $\frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I}_j)}{\partial \mathbf{I}_j}$ is available in any modern deep learning frameworks and the second term $\frac{\partial \mathbf{I}_j}{\partial \boldsymbol{\psi}_j}$ can be obtained from the state-of-the-art differentiable renderer [127]. We can now see more clearly the intuition behind RISP: it requires the network's sensitivity about input images to be orthogonal to the direction that rendering parameters can influence the image, leading to a rendering-invariant prediction.

We stress that the design of this new loss in Eqn. (2.10) is non-trivial. In fact, both $\mathcal{L}^{\text{error}}$ and $\mathcal{L}^{\text{reg}}$ have their unique purposes and must be combined: $\mathcal{L}^{\text{error}}$ encourages $\mathcal{N}$ to fit its output to individually different states, and $\mathcal{L}^{\text{reg}}$ attempts to smooth out its output along the $\boldsymbol{\psi}$ dimension. Specifically, $\mathcal{L}^{\text{reg}}$ cannot be optimized as a standalone loss because it leads to a trivial solution of $\mathcal{N}$ always predicting constant states. Putting $\mathcal{L}^{\text{error}}$ and $\mathcal{L}^{\text{reg}}$ together forces them to strike a balance between predicting accurate states and ignoring noises from

rendering conditions, leading to a network $\mathcal{N}$ that truly learns the "inverse renderer" $\mathcal{R}^{-1}$.

It remains to show how to compute the gradient of the regularizer $\mathcal{L}^{\text{reg}}$ with respect to the network parameters $\boldsymbol{\theta}$, which is required by gradient-based optimizers to minimize this new loss. As the loss definition now includes first-order derivatives, computing its gradients involves second-order partial derivatives, which can be time-consuming if implemented carelessly with multiple loops. Our last contribution is to provide an efficient method for computing this gradient, which can be fully implemented with existing frameworks (PyTorch and `mitsuba`-2 in our experiments):

Theorem 2.3.1. *Assuming forward mode differentiation is available in the renderer $\mathcal{R}$ and reverse mode differentiation is available in the network $\mathcal{N}$, we can compute a stochastic gradient $\frac{\partial \mathcal{L}^{reg}}{\partial \boldsymbol{\theta}}$ in $\mathcal{O}(|\mathbf{s}||\boldsymbol{\theta}|)$ time per image using pre-computed data occupying $\mathcal{O}(\sum_j |\boldsymbol{\psi}_j||\mathbf{I}_j|)$ space.*

In particular, we stress that computing the gradients of $\mathcal{L}^{\text{reg}}$ does *not* require second-order gradients in the renderer $\mathcal{R}$, which would exceed the capability of all existing differentiable renderers we are aware of. We leave the proof of this theorem in our supplemental material.

Further speedup Theorem 2.3.1 states that it takes time linear to the network size and state dimension to compute the gradients of $\mathcal{L}^{\text{reg}}$. The $\mathcal{O}(|\mathbf{s}||\boldsymbol{\theta}|)$ time cost is affordable for very small rigid-body systems (e.g., $|\mathbf{s}| < 10$) but not scalable for larger systems. Therefore, we use a slightly different regularizer in our implementation:

$$\mathcal{L}^{\text{train}}(\boldsymbol{\theta}, \mathcal{D}) = \mathcal{L}^{\text{error}} + \gamma \sum_{(\mathbf{s}_j, \boldsymbol{\psi}_j, \mathbf{I}_j) \in \mathcal{D}} \| \frac{\partial \mathcal{L}_j^{\text{error}}}{\partial \boldsymbol{\psi}_j} \|. \tag{2.12}$$

In other words, we instead encourage the state prediction error to be rendering-invariant. It can be seen from the proof in Theorem 2.3.1 that this new regularizer requires only $\mathcal{O}(\boldsymbol{\theta})$ time to compute its gradients, and we have found empirically that the performance of this new regularizer is comparable to Eqn. (2.10) but is much faster. We leave a theoretical analysis of the two regularizers to future work.

2.4 Experiments

In this section, we conduct various experiments to study the following questions:

1. Q1: Is pixelwise loss on videos across rendering domains sufficient for parameter prediction?
2. Q2: If pixelwise loss is not good enough, are there other competitive alternatives to the state-prediction loss in our approach?
3. Q3: Is the regularizer on rendering gradients in our loss necessary?
4. Q4: How does our approach compare with directly optimizing state discrepancies?
5. Q5: Is the proposed method applicable to real-world scenarios?

We address the first four questions using the simulation environment described in Sec. 2.4.1 and answer the last question using a real-world application at the end of this section. More details about the experiments, including ablation study, can be found in Appendix.

2.4.1 Experimental Setup

Environments We implement four environments (Fig. 2-1): a rigid-body environment without contact (**quadrotor**), a rigid-body environment with contact (**cube**), an articulated body (**hand**), and a deformable-body environment (**rod**). Each environment contains a differentiable simulator [199, 37] and a differentiable renderer [98]. We deliberately generated the training set in Sec. 2.3 using a different renderer [127] and used different distributions when sampling rendering configurations in the training set and the environments.

Tasks We consider four types of tasks defined on the physical systems in all environments: state estimation, system identification, imitation learning, and visuomotor control. The state estimation task (Sec. 2.4.2) require a model to predict the state of the physical system from a given image and serves as a prerequisite for the other downstream tasks. The system identification (Sec. 2.4.3) and imitation learning (Sec. 2.4.4) tasks aim to recover the system parameters and control signals of a physical system from the video, respectively. Finally, in

the visuomotor control task (Appendix), we replace the video with a target image showing the desired state of the physical system and aim to discover the proper control signals that steer the system to the desired state. In all tasks, we use a photorealistic renderer [138] to generate the target video or image.

Baselines We consider two strong baselines: The **pixelwise-loss** baseline is used by ∇`Sim` [126], which is the state-of-the-art method for identifying system parameters directly from video inputs. We implement ∇`Sim` by removing RISP from our method and back-propagating the pixelwise loss on images through differentiable rendering and simulation. We run this baseline to analyze the limit of pixelwise loss in downstream tasks (Q1). The second baseline is **preceptual-loss** [79], which replaces the pixelwise loss in ∇`Sim` with loss functions based on high-level features extracted by a pre-trained CNN. By comparing this baseline with our method, we can justify why we choose to let RISP predict states instead of other perceputal features (Q2).

We also include two weak baselines used by ∇`Sim` : The **average** baseline is a deterministic method that always returns the average quantity observed from the training set, and the **random** baseline returns a guess randomly drawn from the data distribution used to generate the training set. We use these two weak baselines to avoid designing environments and tasks that are too trivial to solve.

Our methods We consider two versions of our methods in Sec. 2.3: **ours-no-grad** implements the domain randomization idea without using the proposed regularizers, and **ours** is the full approach that includes the regularizer using rendering gradients. By comparing between them, we aim to better understand the value of the rendering gradients in our proposed method (Q3).

Oracle Throughout our experiments, we also consider an oracle method that directly minimizes the state differences obtained from simulation outputs without further rendering. In particular, this oracle sees the ground-truth states for each image in the target video or image, which is inaccessible to all baselines and our methods. We consider this approach to be an oracle because it is a subset of our approach that involve differentiable physics only,

but it needs a perfect state-prediction network. This oracle can give us an upper bound for the performance of our method (Q4).

Training We build our RISP network and baselines upon a modified version of ResNet-18 [60] and train them with the Adam optimizer [84]. We report more details of our network architecture, training strategies, and hyperparameters in Appendix.

2.4.2 State Estimation Results

In this task, we predict the states of the physical system from randomly generated images and report the mean and standard deviation of the state prediction errors in Table 2.1. Note that we exclude the **perceptual-loss** and **pixelwise-loss** baselines as they do not require a state prediction step. Overall, we find that the state estimation results from our methods are significantly better than all baselines across the board. The two weak baselines perform poorly, confirming that this state-estimation task cannot be solved trivially. We highlight that our method with the rendering-gradient loss predicts the most stable and accurate state of the physical system across the board, which strongly demonstrates that RISP learns to make predictions independent of various rendering configurations.

2.4.3 System Identification Results

Our system identification task aims to predict the system parameters of a physical system, e.g., mass, density, stiffness, or elasticity, by watching a reference video with known action sequences. For each environment, we manually design a sequence of actions and render

	quadrotor	**cube**	**hand**	**rod**
average	0.5994 ± 0.0000	0.5920 ± 0.0000	0.2605 ± 0.0000	0.9792 ± 0.0000
random	0.9661 ± 0.7548	0.9655 ± 0.8119	0.8323 ± 0.5705	1.2730 ± 0.7197
ours-no-grad	0.3114 ± 0.3191	0.2805 ± 0.3199	0.1155 ± 0.0505	0.0201 ± 0.0087
ours	$\mathbf{0.1505 \pm 0.1163}$	$\mathbf{0.1642 \pm 0.1887}$	$\mathbf{0.0974 \pm 0.0255}$	$\mathbf{0.0194 \pm 0.0048}$

Table 2.1: State estimation results (Sec. 2.4.2). Each entry in the table reports the mean and standard deviation of the state estimation error computed from 800 images under 4 rendering configurations.

a reference video of its dynamic motion. Next, we randomly pick an initial guess of the system parameters and run gradient-based optimization using all baselines, our methods, and the oracle. We repeat this experiment 4 times with randomly generated rendering conditions and initial guesses and report the mean and standard deviation of each system parameter in Table 2.2. The near-perfect performance of the oracle suggests that these system identification tasks are feasible to solve as long as a reliable state estimation is available. Both of our methods outperform almost all baselines by a large margin, sometimes even by orders of magnitude. This is as expected, since the previous task already suggests that our methods can predict states from a target video much more accurate than baselines, which is a crucial prerequisite for solving system identification. The only exception is in the **cube** environment, where the pixelwise loss performs surprisingly well. We hypothesize it may be due to its relatively simple geometry and high contrast from the background (Fig. 2-1).

2.4.4 Imitation Learning Results

Our imitation learning tasks consider the problem of emulating the dynamic motion of a reference video. The experiment setup is similar to the system identification task except that we swap the known and unknown variables in the environment: The system parameters are now known, and the goal is to infer the unknown sequence of actions from the reference video. Note that the **cube** environment is excluded because it has no control signals. As before, we repeat the experiment in all environments 4 times with randomly generated

	quadrotor mass	**cube** stiffness	**hand** joint stiffness	**rod** Young's modulus
random	9.22e-2±3.83e-2	2.31e-1±9.57e-2	5.70e-1±1.62e-1	1.30e6±1.35e6
pixelwise-loss	7.22e-2±5.26e-2	**2.24e-3±1.74e-3**	1.04e-1±9.62e-2	8.50e5±1.42e6
perceptual-loss	6.45e-2±5.23e-2	1.16e-1±5.83e-2	1.10e-1±1.18e-1	8.32e5±1.43e6
ours-no-grad	6.07e-2±4.34e-2	1.16e-1±6.20e-2	4.85e-2±**2.16e-2**	8.78e4±1.52e5
ours	**1.18e-2±1.93e-2**	6.76e-3±7.23e-3	**3.96e-2**±2.73e-2	**9.31e1±4.21e1**
oracle	2.36e-5±2.41e-5	1.15e-3±8.60e-4	3.92e-3±4.40e-4	4.36e0±3.57e0

Table 2.2: System identification results (Sec. 2.4.3). Each entry reports the mean and standard deviation of the parameter estimation error computed from 4 random initial guesses and rendering conditions.

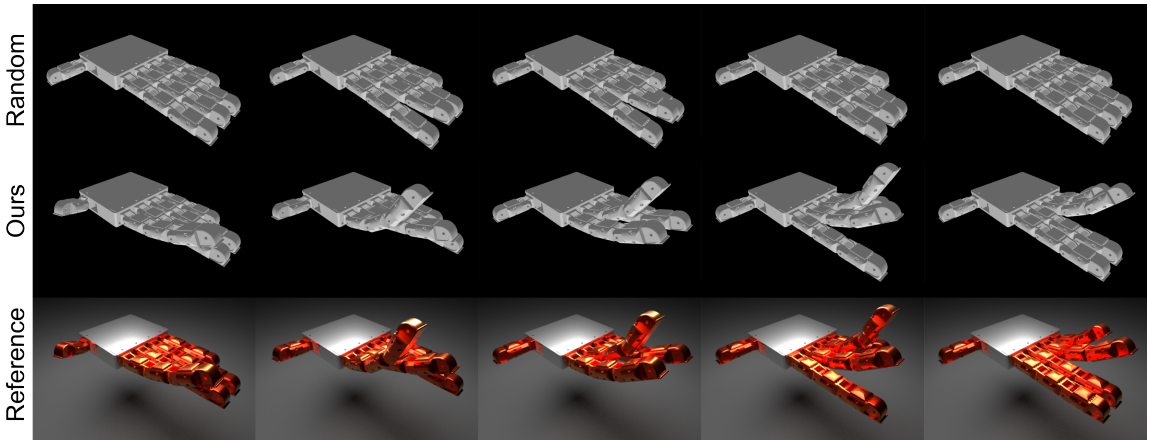


Figure 2-3: Imitation learning in the hand environment (Sec. 2.4.4). Given a reference video (bottom row, shown as five intermediate frames), the goal is to reconstruct a sequence of actions that resembles its motion. We show the motions generated using a randomly chosen initial guess of the actions (top row) and optimized actions using our method with rendering gradients (middle row).

rendering configurations and initial guesses of the actions. We report the results in Table 2.3 and find that our method with rendering gradients (**ours** in the table) achieves much lower errors, indicating that we resemble the motions in the video much more accurately. The errors from **pixelwise-loss** have smaller variations across rendering domains but are larger than ours, indicating that it struggles to solve this task under all four rendering configurations. In addition, the oracle finds a sequence of actions leading to more similar motions than our method, but it requires full and accurate knowledge of the state information which is rarely accessible from a video. We visualize our results in the **hand** environment in Fig. 2-3.

	quadrotor	**hand**	**rod**
average	28.40 ± 0.00	7.62 ± 0.00	30.20 ± 0.00
random	1120 ± 113	10.27 ± 0.91	29.89 ± 0.61
pixelwise-loss	$12.65 \pm \mathbf{0.13}$	$6.71 \pm \mathbf{0.07}$	29.57 ± 0.85
perceptual-loss	N/A	5.10 ± 1.80	14.61 ± 5.13
ours-no-grad	25.07 ± 5.76	7.12 ± 0.71	$\mathbf{0.83} \pm 0.35$
ours	$\mathbf{2.63} \pm 1.86$	$\mathbf{1.52} \pm 0.18$	$1.05 \pm \mathbf{0.27}$
oracle	0.79 ± 0.44	0.02 ± 0.02	0.28 ± 0.16

Table 2.3: Imitation learning results (Sec. 2.4.4). Each entry reports the mean and standard deviation of the state discrepancy computed from 4 randomly generated initial guesses and rendering conditions. N/A indicates failure of convergence after optimization.

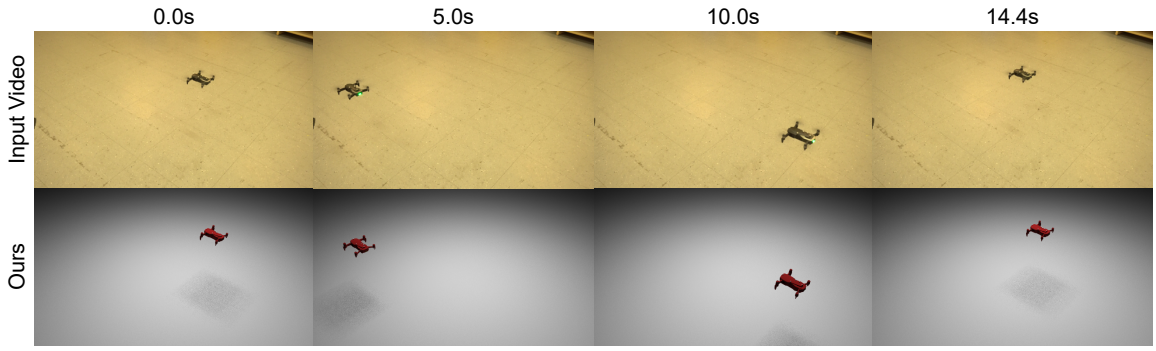


Figure 2-4: Imitation learning in the real-world experiment (Sec. 2.4.5). Given a reference video (top row), the goal is to reconstruct a sequence of actions sent to a virtual quadrotor that resembles its motion. We illustrate the motions reconstructed using our method (bottom row).

2.4.5 A Real-World Application

Finally, we evaluate the efficacy of our approach on a real-world application: Given a video of a flying quadrotor, we aim to build its digital twin that replicates the video motion by inferring a reasonable sequence of actions. This real-to-sim transfer problem is challenging due to a few reasons: First, the real-world video contains complex textures, materials, and lighting conditions unknown to us and unseen by our differentiable renderer. Second, the quadrotor's real-world dynamics is polluted by environmental noises and intricate aerodynamic effects, which are ignored in our differentiable simulation environment. Despite these challenges, our method achieved a qualitatively good result with a standard differentiable rigid-body simulator and renderer, showing its generalizability across moderate discrepancies in dynamic models and rendering configurations (Fig. 2-4). Our approach outperforms all baselines by a large margin in this task, which we detail in Appendix.

2.5 Conclusions, Limitations, and Future Work

We proposed a framework that integrates rendering-invariant state-prediction into differentiable simulation and rendering for cross-domain parameter estimation. The experiments in simulation and in real world have shown that our method is more robust than pixel-wise or perceptual losses on unseen rendering configurations. The additional ablated study further confirms the efficiency and efficacy comes from using the rendering gradients in our RISP

network.

Despite its promising results, RISP still has a few limitations. Firstly, we require a differentiable simulator that can capture the dynamic model of the object, which may not be available for real-world scenes with intricate dynamics, e.g., fluids. A second limitation is that our method requires knowledge of camera parameters, which is not always accessible in a real-world scenario. This limitation can be resolved by incorporating in RISP the gradients for camera parameters available in modern differentiable renderers. Lastly, we assume a moderately accurate object geometry is available, which can potentially be relaxed by combining NeRF [122] with our approach to infer the geometry.

Chapter 3

Learning Neural Constitutive Laws from Motion Observations for Generalizable PDE Dynamics

3.1 Introduction

Partial-differential-equation-governed dynamical systems are ubiquitous in physical, chemical, and biological settings [52]. Computationally solving these PDEs, e.g., using the finite element method (FEM) [71], has enabled critical applications in science and engineering. Recently, research communities have explored the role of machine learning (ML) in advancing partial differential equation (PDE) modeling [81]. Some methods, e.g., physics-informed neural network (PINN) [146], assume the entire PDE is known. Other methods, e.g., graph neural network (GNN) [154, 137], do not assume any knowledge about the underlying PDE. These methods learn the PDE-governed dynamics entirely from data. Without any knowledge about the PDE, these approaches are prone to violating physical laws and have limited generalizability. In this work, we argue that instead of learning the entire PDE with neural solutions, there are benefits of replacing only *parts* of the PDEs with neural components while keeping the rest of the PDEs intact. For example, we contend that commonly-agreed physical laws, such as the elastodynamics equation [50], do not need to

be learned. Instead, these commonly-agreed physical laws should be explicitly enforced to guarantee physical correctness. By contrast, we champion that those parts of the PDEs where researchers traditionally struggle to model can particularly benefit from a learning solution. In particular, we explore an ML approach for an indispensable part of many PDEs: the constitutive laws.

A constitutive law describes the relationship between two physical quantities [175], e.g., stress and strain in solids [49], shear stress and shear rate in fluids [174], the electric displacement field, and the electric field in electromagnetism [92]. These constitutive laws play crucial roles in important PDEs, such as the elastodynamic, Navier-Stokes, and Maxwell's equations.

Traditionally, constitutive laws are manually designed by domain experts. While they should satisfy generally-accepted constraints, e.g., frame indifference (rotational equivariance), the ultimate criterion of good constitutive laws is how well they match the experimentally observed data, which is often highly nonlinear. As such, to design constitutive relationships, researchers have relied on various data-fitting tools, ranging from high-order polynomials [128] and splines [198] to exponential models [61]. Recently, neural-network-based models have been shown to achieve higher accuracies than their classic, expert-constructed counterparts, thanks to the neural network's large learning capacity [48, 94]. Furthermore, these models alleviate the need to manually design function forms by hand since the same network architecture can be used to capture various mechanical behaviors [108]. Indeed, we show in our work that a single neural network architecture can capture diverse constitutive behaviors, ranging from water to elastic solids, which previously required case-by-case, expert-designed constitutive models.

When it comes to learning constitutive laws via neural networks, two challenges stand out. First, how to enforce these neural constitutive laws to obey known physics priors, e.g., frame indifference? Second, how to obtain the labeled data for supervised training? For example, in the case of learning elasticity, one must obtain labeled stress-strain pairs [8, 38, 44]. However, these stress-strain pairs are often either impossible to measure, e.g., *in vivo* medical imaging [1], or require highly specialized devices [14].

To address the first question, we introduce the physical priors as an inductive bias in

the network architecture and directly bake in these priors *by design*. In this way, our network naturally satisfies the necessary priors without tons of data augmentation. To address the challenge of labeled data, we embed the learnable constitutive model inside a differentiable PDE-based physical simulator [126] and do supervised training on the output of this physical simulator, *not* on the output of the constitutive model itself. In particular, our approach trains on motion observation data (formally defined as kinematic information in the continuum mechanics literature), e.g., material point positions, which are significantly easier to measure than stress information.

With the proposed constitutive learning approach, we demonstrate generalizable PDE dynamics on which pure NN approaches, e.g., GNN, struggle. These generalization tasks include extreme out-of-distribution generalization over temporal ranges, initial/boundary conditions, geometries, and multi-physics systems while training on *a single motion trajectory*. We attribute our generalization advantage over pure NN approaches to the fact that we only learn one element of the PDE while keeping the PDE's widely-accepted elements intact (e.g., conservation of mass and momentum).

In summary, our work makes the following contributions:

- We embed a learnable constitutive law inside a differentiable physics simulator and train the constitutive law directly on the simulator's output.

- We guarantee the satisfaction of physical priors to constitutive laws by designing network architectures with matrix-wise rotation equivariance and undeformed state equilibrium.

- We validate our hybrid NN-PDE approach on large-deformation PDE dynamics featuring one-shot generalization over unseen temporal ranges, initial conditions, boundary conditions, geometries, and multi-physics systems, on which previous pure NN approaches fail.

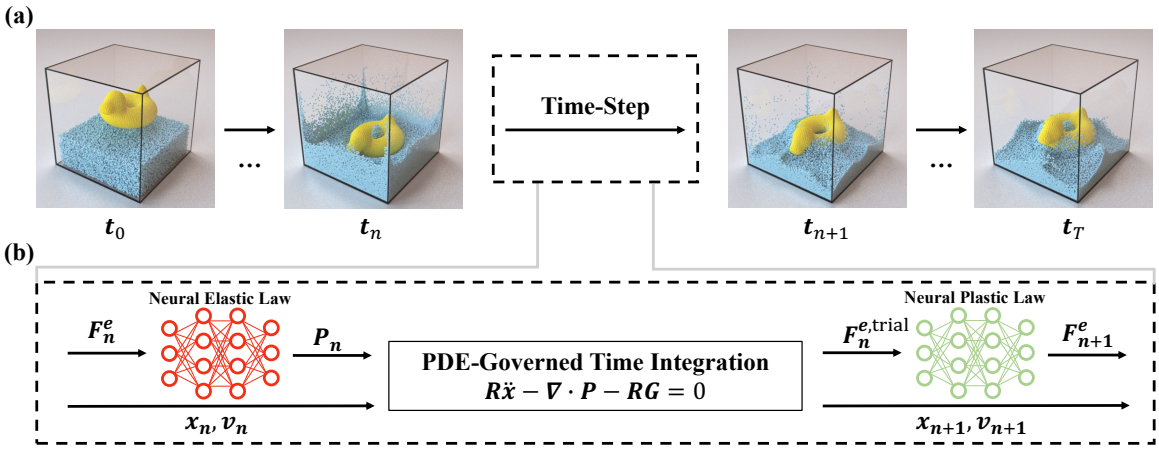


Figure 3-1: Hybrid NN-PDE time-stepping. (a) Our method "time-steps" sequentially to obtain the solution of the dynamical system (b) Inside the time-stepping algorithm, we (i) use the neural elastic constitutive law to obtain the stress, (ii) update the state by solving the governing PDE (Eqn. (3.1)), and (iii) obtain the new elastic deformation gradient via the neural plastic constitutive law. Algorithm 2 lists the corresponding pseudocode.

3.2 Related Work

Constitutive Laws Constitutive laws date back to 1676 when Robert Hooke, F.R.S. observed the linear relationship between spring force and deformation, i.e., $F_s = kx$ [170]. Since then, constitutive laws have been the staple of various branches of physics: elasticity [172, 43, 7], plasticity [123, 34], fluids [26], and permittivity [92]. While these constitutive laws have traditionally been constructed through nonlinear polynomial bases [198, 197], research communities have been embracing neural networks due to their versatility and robustness [158, 169, 190, 186, 185, 42, 166, 184, 183, 87, 106]. Most of these neural-network-based approaches require labeled data for the input and output of the constitutive model. By contrast, our work does not require any such data. Instead, we embed the neural constitutive laws inside a differentiable-simulation-based training pipeline.

Differentiable Simulation implements traditional PDE simulations (e.g., FEM) in a differentiable manner such that the simulation can be employed in a gradient-based optimization workflow [55, 104, 114, 66, 67, 70, 37, 36, 142, 141, 35, 30, 47, 196, 32]. Notably, these differentiable physics simulations enable the recovery of constitutive parameters directly from videos and motion observations [126, 113, 22], e.g., Young's modulus and Poisson's

ratio of elasticity. Prior differentiable simulation works assume that an expert-designed constitutive law is known *a priori*. However, given a motion observation sequence of an arbitrary material, it is a non-trivial task for the practitioners to identify the underlying constitutive model, not to mention recovering their parameters. By contrast, we do not assume the availability of the material model. Instead, we parameterize the constitutive laws via generally-applicable neural network architectures. Wang et al. [188] and Huang et al. [68] also demonstrate promising results in training neural constitutive laws directly from motion observations, but their formulations are limited to isotropic elastic materials and 2D quasi-static loadings, respectively. Our generic formulation handles both isotropic and anisotropic materials as well as time-dependent 3D elastoplastic dynamics.

Machine Learning (ML) for PDE Dynamics Our work falls into the broader conversations on ML for PDEs. Physics-informed neural networks (PINNs) [146] represent the PDE solution via a neural representation and introduce physics priors via PDE-informed loss. DeepONet [112] and neural operator [103] learn mappings from the problem parameters to the solution with the goal of achieving fast surrogate models. Unlike these approaches, our work operates *entirely within* the classical numerical solver framework. We propose an ML solution for *one and only one component* of the classical solver: the constitutive model of the PDE. There are many great works that only replace one part of the PDE with the neural network, although not in a constitutive law context. For example, Bar-Sinai et al. [10]'s data-driven discretizations, Yin et al. [204]'s APHYNITY, and Um et al. [178]'s solver-in-the-loop approach. Another robust ML-PDE framework is the graph neural network (GNN). GNN directly learns how much force is applied to a discretized particle [154, 137] at every time step. Implicitly, GNN learns *both* the constitutive law (material model) *and* the equation of motion (i.e., the elastodynamics equation). As such, our work differs from GNN by incorporating the equation-of-motion prior, which holds true in standard continuum mechanics settings [50], and only treating the constitutive model with an NN approach. Relatedly, Li et al. [100] augment classic numerical solvers with neural-work-based plasticity energy, aiming to improve optimization-time-integrators. By contrast, our work focuses on constitutive laws, not time integrators.

3.3 Method

Given a motion sequence of materials undergoing deformations, our goal is to obtain their constitutive models, represented via neural networks. These constitutive laws can later be used to predict scenarios unseen in the training motion sequence. Sec. 3.3.1 first recaps the current state-of-the-art framework for identifying classic constitutive parameters from motions observations. Secs. 3.3.2 and 3.3.3 will introduce the proposed neural alternative.

3.3.1 Background

Continuous PDE In this work, we assume all materials in the experiments observe the elastodynamic equation [49]:

$$\rho_0 \ddot{\boldsymbol{\phi}} = \nabla \cdot \boldsymbol{P} + \rho_0 \boldsymbol{b}. \tag{3.1}$$

The vector field of interest is the deformation map $\boldsymbol{\phi}$, which uniquely describes the current position $\boldsymbol{x}$ of an arbitrary spatial point with an initial position $\boldsymbol{X}$ at time t, i.e., $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{X}, t)$. Additionally, ρ_0 is the initial density, $\boldsymbol{P} = \widehat{\boldsymbol{P}}(\boldsymbol{F}^e)$ is the first Piola-Kirchhoff stress, $\boldsymbol{F}^e$ is the elastic part of the deformation gradient ($\nabla\boldsymbol{\phi}$), $\dot{\boldsymbol{\phi}}$ and $\ddot{\boldsymbol{\phi}}$ are the velocity and acceleration, and $\boldsymbol{b}$ is the body force. In principle, our approach can also be extended to other PDEs involving spatiotemporal gradients, e.g., the Navier-Stokes equations.

Constitutive laws For Eqn. (3.1) to be well-defined, constitutive relationships must be prescribed. The elastic constitutive law defines the relationship between $\boldsymbol{P}$ and $\boldsymbol{F}^e$: $\boldsymbol{P} = \widehat{\boldsymbol{P}}(\boldsymbol{F}^e)$. The plastic constitutive law defines an inequality constraint: $f_Y(\boldsymbol{P}) < 0$ (where f_Y is the yield function [16]), as well as a flow rule when this constraint is violated. Constitutive laws in these forms have been shown to capture a wide range of materials: from elastic solids to granular media.

Discretization In order to numerically solve Eqn. (3.1), we discretize it spatially and temporally, yielding a dynamical system $\mathcal{M}$:

$$\mathbf{s}_{n+1} = \mathcal{M}_\mu(\mathbf{s}_n), \forall n = 0, 1, \ldots, T, \tag{3.2}$$

where T is the total number of time steps in the simulation, and $\mathbf{s}_n$ and $\mathbf{s}_{n+1}$ are the state vectors at the corresponding time steps. The vector μ encodes all the system parameters, e.g., those of the constitutive laws.

To obtain the dynamical system $\mathcal{M}$, we may use any discretization technique, e.g., FEM, finite volume methods [39], smoothed-particle hydrodynamics [124]. In this work, we adopt the material point method (MPM) [164, 76, 65] for its versatility in handling various materials. With MPM, we discretize the domain with Q material points, and $\mathcal{M}_\mu$ becomes Algorithm 1. Here, the state vector $\mathbf{s}_n = \{\boldsymbol{x}_n, \boldsymbol{b}_n, \boldsymbol{F}_n^e\}$ contains all the discretized material points' current positions, velocities, and elastic deformation gradients. The time integration scheme $\mathcal{I}$ follows standard MPM practices, derived from the weak form of Eqn. (3.1). Appendix B.1 provides additional background on MPM.

We highlight that the main contribution of our work is orthogonal to the choice of discretization. The time integration scheme $\mathcal{I}$ can also be replaced by other PDE-governed, weak-form-derived numerical methods, e.g., FEM. Our main contribution is the *continuous* constitutive law, which is required by all numerical methods.

Indeed, the time-stepping scheme involves two constitutive laws (Eqn. (3.3)):

$$\begin{aligned} \mathcal{E}_\mu &: \boldsymbol{F}^e \mapsto \boldsymbol{P} & \mathcal{P}_\mu &: \boldsymbol{F}^{e,\text{trial}} \mapsto \boldsymbol{F}^{e,\text{new}} \\ &: \mathbb{R}^{3\times3} \to \mathbb{R}^{3\times3} & &: \mathbb{R}^{3\times3} \to \mathbb{R}^{3\times3}. \end{aligned} \tag{3.3}$$

Algorithm 1 Time-stepping, classic

Input: $\mathbf{s}_n = \{\boldsymbol{x}_n, \boldsymbol{b}_n, \boldsymbol{F}_n^e\}$
Output: $\mathbf{s}_{n+1} = \{\boldsymbol{x}_{n+1}, \boldsymbol{b}_{n+1}, \boldsymbol{F}_{n+1}^e\}$
1: for $i = 1 \ldots Q$, $\boldsymbol{P}_n(i) = \mathcal{E}_\mu(\boldsymbol{F}_n^e(i))$
2: $\boldsymbol{x}_{n+1}, \boldsymbol{b}_{n+1}, \boldsymbol{F}_{n+1}^{e,\text{trial}} = \mathcal{I}(\boldsymbol{x}_n, \boldsymbol{b}_n, \boldsymbol{F}_n^e, \boldsymbol{P}_n)$
3: for $i = 1 \ldots Q$, $\boldsymbol{F}_{n+1}^e(i) = \mathcal{P}_\mu(\boldsymbol{F}_{n+1}^{e,\text{trial}}(i))$

The elastic constitutive law $\mathcal{E}_\mu$ computes the first Piola-Kirchhoff stresses $\boldsymbol{P}_n$ from the elastic deformation gradients $\boldsymbol{F}_n^e$. The plastic return-mapping scheme $\mathcal{P}_\mu$ [31] projects the trial elastic deformation gradient onto the plastic yield constraint f_Y. Examples of $\mathcal{E}_\mu$ and $\mathcal{P}_\mu$ are listed in Appendix B.2.

System Identification from Motion Observations Once we have the PDE-governed simulation, we can identify constitutive parameters directly from motion observations [113]. In particular, we can define the loss function on the ground-truth observed particle positions $\boldsymbol{x}_n^{\text{gt}}$:

$$\min_{\mu} \mathcal{L}(\{\boldsymbol{x}_n^{\text{gt}}\}_{n=0}^T, \{\boldsymbol{x}_n\}_{n=0}^T), \tag{3.4}$$

where $\mathcal{L}$ is the loss measure. By implementing the simulation (Algorithm 1) in a differentiable manner, we can employ standard gradient-based optimization techniques to solve Eqn. (3.4) and optimize for the constitutive parameters that match the motion observations.

3.3.2 Neural Constitutive Laws

In this work, we replace the classic elastic and plastic constitutive treatments with neural-network-based models: $\mathcal{E}_{\theta_e}$ and $\mathcal{P}_{\theta_p}$, where θ_e and θ_p are the neural network weights. The time-stepping scheme now becomes Algorithm 2. Fig. 3-1 provides a schematic illustration of our hybrid NN-PDE time-stepping scheme. Notably, we only represent the constitutive laws with neural networks while keeping the classic PDE-based time integration $\mathcal{I}$ unchanged. The time integration scheme $\mathcal{I}$ computes the force from the divergence of stress and updates the particle positions and velocities via standard Euler methods [9]. Since these steps follow well-known physical laws, we argue that they do not need to be

Algorithm 2 Time-stepping, neural

Input: $\mathrm{s}_n = \{\boldsymbol{x}_n, \boldsymbol{b}_n, \boldsymbol{F}_n^e\}$
Output: $\mathrm{s}_{n+1} = \{\boldsymbol{x}_{n+1}, \boldsymbol{b}_{n+1}, \boldsymbol{F}_{n+1}^e\}$
1: for $i = 1 \dots Q$, $\boldsymbol{P}_n(i) = \mathcal{E}_{\theta_e}(\boldsymbol{F}_n^e(i))$
2: $\boldsymbol{x}_{n+1}, \boldsymbol{b}_{n+1}, \boldsymbol{F}_{n+1}^{e,\text{trial}} = \mathcal{I}(\boldsymbol{x}_n, \boldsymbol{b}_n, \boldsymbol{F}_n^e, \boldsymbol{P}_n)$
3: for $i = 1 \dots Q$, $\boldsymbol{F}_{n+1}^e(i) = \mathcal{P}_{\theta_p}(\boldsymbol{F}_{n+1}^{e,\text{trial}}(i))$

learned. To train on motion observation data, we also adopt the same objective function as Eqn. (3.4), but the optimized variable now becomes the neural network weights:

$$\min_{\theta_e,\theta_p} \mathcal{L}(\{\boldsymbol{x}_n^{\text{gt}}\}_{n=0}^T, \{\boldsymbol{x}_n\}_{n=0}^T). \tag{3.5}$$

We also emphasize that we do not assume a general backbone form for the constitutive law. Assuming a particular backbone (e.g., neo-Hookean elasticity) would prevent us from capturing the constitutive behaviors of another one. Therefore, we opt not to assume any backbone and represent the entire constitutive law with NNs, which are data-driven general representations with better expressiveness and versatility. However, the framework outside constitutive laws can be considered a general PDE-based backbone. In this sense, we share similar philosophy as Yin et al. [204] by augmenting physical models with NNs.

3.3.3 Physics-Aware Network Architecture

Our neural constitutive laws must satisfy essential physics priors [187]. Nevertheless, we also avoid being constrained by any overly strong prior, e.g., isotropic prior, which prevents our model from capturing anisotropic materials [188]. In this work, we consider two generally applicable physical priors.

Rotation Equivariance The properties of most materials in the physical world remain invariable under rotations. This property is also known as frame indifference. Unlike previous works [33] focusing on vector data (e.g. point cloud), we have to enforce the rotation equivariance of deformation gradients, which are square matrices. To this end, we feed the neural networks with a selected set of rotation invariants, including the principal stretches $\boldsymbol{\Sigma}$ (singular values), the right Cauchy-Green tensor $\boldsymbol{F}^{eT}\boldsymbol{F}^e$, and the determinant of the elastic deformation gradient $\det(\boldsymbol{F}^e)$. Then, we take the rotation matrix $\boldsymbol{R}$ out of the deformation gradient via polar decomposition $\boldsymbol{F}^e = \boldsymbol{RS}$ and multiply it back to the output of the neural networks to ensure the rotation equivariance. Appendix B.3 provides the detailed algorithm and proofs for rotation invariance of the neural networks and neural constitutive laws.

Table 3.1: Reconstruction. We show the reconstruction losses for different methods. For each environment, we indicate the top 1 with dark green and the top 2 with light green from all the baselines and our method. We also report the statistics (mean ± standard derivation) for each method in the last column.

Method	Jell-O	Sand	Plasticine	Water	**Overall**
spline	2.4e-1	3.2e-1	3.0e-1	3.2e-1	2.9e-1±3.5e-2
neural	1.2e-5	1.2e-2	1.9e-1	2.9e-2	5.7e-2±8.7e-2
gnn	2.1e-2	1.1e-2	8.7e-3	3.2e-2	1.8e-2±1.0e-2
ours-init	1.1e-1	4.8e-2	4.6e-2	6.8e-2	6.8e-2±2.8e-2
ours	2.4e-4	2.6e-5	6.5e-5	2.0e-5	8.6e-5±1.0e-4
labeled	8.0e-9	7.1e-2	5.5e-6	1.6e-7	1.8e-2±3.5e-2
sys-id	1.7e-8	2.7e-7	5.8e-10	1.7e-8	7.6e-8±1.3e-7

Undeformed State Equilibrium Another important constitutive prior is preserving the rest shape under zero load. We ensure the undeformed state equilibrium by (1) normalizing the input invariants so that they are zero under zero load and (2) removing all bias layers from the neural networks. Therefore, when zero loads are applied, the elasticity network will generate zero stress, while the plasticity network will preserve the trial elastic deformation gradient.

3.4 Experiments

In this section, we first show our experimental setup (Sec. 3.4.1). We then compare our methods against baselines and oracles in reconstruction (Sec. 3.4.2) and generalization (Sec. 3.4.3). We further study our method in two advanced experiments: the multi-physics environments (Sec. 3.4.5) and a real-world experiment (Sec. 3.4.6). We refer the readers to our project website[1] where the results are best illustrated in videos.

3.4.1 Experimental Setup

Environments We consider four distinct elastoplastic materials covering a diverse set of physical effects: purely elastic (Jell-O), elastic with the Drucker-Prager yield condition

[1] https://sites.google.com/view/nclaw

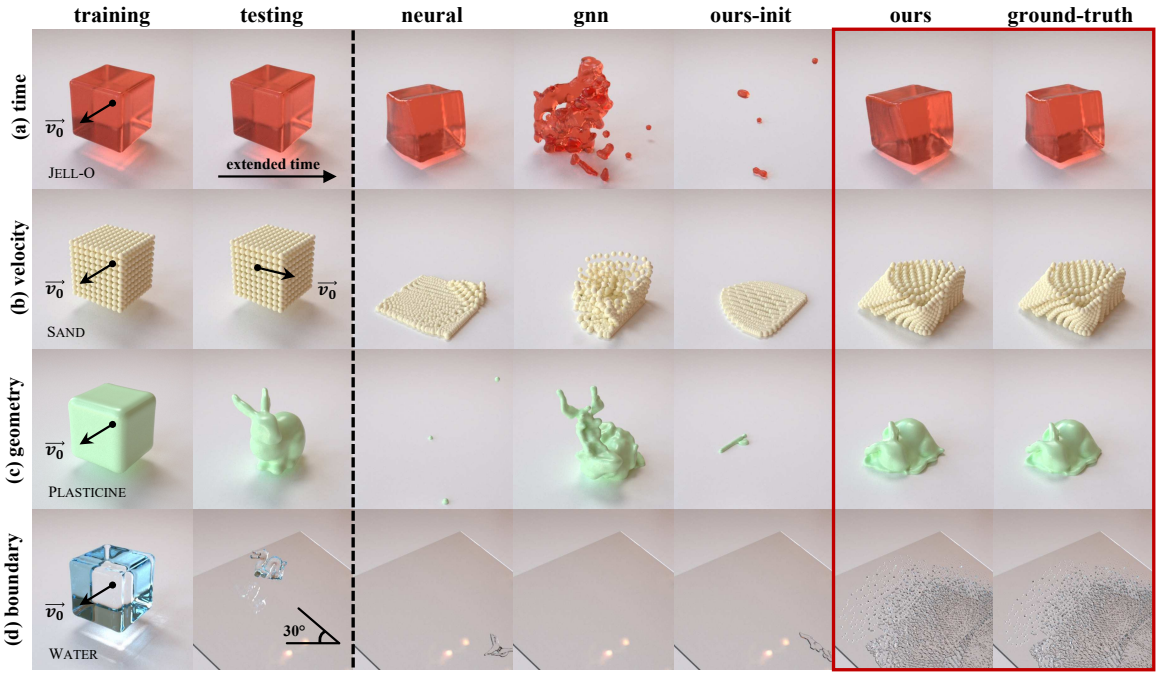


Figure 3-2: Generalization. We first train all methods in the environments specified in training with the initial velocities indicated using the black arrows. Then, we evaluate the generalization of all methods on four tasks specified in testing: (a) extended time, (b) unseen initial velocity, (c) challenging geometry, and (d) inclined boundary. The left side of the dashed line shows the initialization, and the right side shows the simulated results after a period of time. We compare our method (ours) against baselines (neural and gnn), our initialization (ours-init), and the ground-truth simulation (ground-truth). We highlight ours and ground-truth with a red box.

(Sand), elastic with the von Mises yield condition (Plasticine), and weakly compressible fluids (Water). We detail the mathematical definition and implementation of these ground-truth material models in Appendix B.2.

Baselines We compare our method against three strong baselines. spline [198] models the constitutive laws of the elasticity using Bézier splines, where the control points are the parameters for optimization. neural [188] builds upon spline but replaces the Bézier spline with neural networks. gnn [154] learns the particle dynamics with graph neural networks without any assumption about underlying PDEs.

Our Methods In addition to our results after a full training (ours), we also present the performance of our method at the initial stage (ours-init) without training using Eqn. (3.5)

Table 3.2: Generalization. We show the generalization losses for different methods. We report the quantitative results on three main tasks from Fig. 3-2: (a) extended time, (b) unseen initial velocity, and (c) challenging geometry. Note that we evaluate with three random initial velocities for task (b) and report the average loss for more thorough validation. For each task in each environment, we indicate the top 1 with dark green and the top 2 with light green from all the baselines and our method. We also report the statistics (mean $\pm$ standard derivation) for each method in the last column.

Method	Jell-O			Sand			Plasticine			Water			**Overall**
	(a)	**(b)**	**(c)**	**(a)**	**(b)**	**(c)**	**(a)**	**(b)**	**(c)**	**(a)**	**(b)**	**(c)**	
spline	2.6e-1	2.4e-1	3.5e-1	3.5e-1	3.1e-1	3.6e-1	3.0e-1	3.0e-1	3.1e-1	3.7e-1	3.2e-1	3.3e-1	3.1e-1±4.0e-2
neural	2.9e-5	1.4e-5	5.6e-5	4.7e-2	6.2e-3	2.3e-1	2.4e-1	4.1e-3	2.8e-1	4.6e-2	1.7e-2	5.1e-2	4.9e-2±8.8e-2
gnn	3.3e-2	1.5e-2	1.6e-1	1.5e-2	1.7e-2	3.4e-1	1.1e-2	6.8e-3	3.7e-2	1.1e+0	5.8e-2	2.9e-1	1.2e-1±2.6e-1
ours-init	1.5e-1	6.6e-2	1.2e-1	1.5e-1	2.7e-2	5.0e-2	1.3e-1	2.7e-2	5.8e-2	3.0e-1	4.1e-2	1.0e-1	7.7e-2±6.9e-2
ours	9.8e-4	2.4e-4	4.1e-4	4.2e-5	6.5e-5	3.6e-4	1.4e-4	4.6e-5	2.3e-4	3.5e-4	1.9e-5	2.4e-4	1.9e-4±2.3e-4
labeled	1.1e-7	7.8e-8	1.7e-4	2.2e-1	1.8e-3	3.6e-1	1.5e-5	7.5e-6	1.1e-4	2.1e-6	2.2e-7	1.8e-6	2.9e-2±9.2e-2
sys-id	4.0e-8	6.1e-9	1.1e-8	3.7e-7	4.4e-8	1.6e-7	7.2e-10	1.3e-10	1.4e-9	2.6e-7	7.2e-9	1.4e-8	5.1e-8±9.9e-8

in order to emphasize the efficacy of learning.

Oracles We consider two oracles with extra knowledge that is normally inaccessible. labeled [8] utilizes labeled ground truth of $\boldsymbol{P}$ and $\boldsymbol{F}^{e,\text{new}}$ for supervision instead of the motion observations. sys-id [113] assumes a given constitutive model and identifies only the material parameters (e.g., Young's modulus).

Implementation Details For training, we load a cubic object with 1k homogeneous material points at the center of a box with a fixed linear and angular velocity. We simulate the cube's motion for a total of 1k time steps with a step size of 5e-4s. We generate *one single* trajectory for each environment as the training dataset. For our neural networks, we use a 3-layer multilayer perceptron (MLP) with 64 neurons per layer. Our quantitative results report the average mean square error every 5 frames since gnn is trained with 5$\times$ step size following [154]. We provide more implementation details in Appendix B.4.

3.4.2 Reconstruction

We first train all methods on a single trajectory in each environment to reconstruct the motion sequence. We quantitatively demonstrate the comparison of the reconstruction losses in Tab. 3.1. We find spline ends with constantly poor results in all environments, likely be-

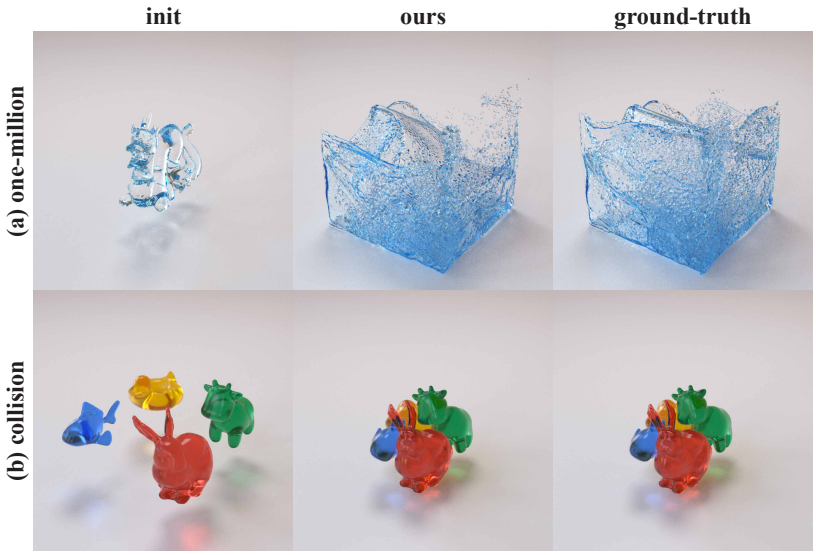


Figure 3-3: Extreme Generalization. We compare the initial state, our method, and the ground-truth results from the traditional simulation in two extreme generalization experiments: (a) a complex water dragon loaded with over 1 million particles, (b) four rubber toys vigorously colliding with each other and the floor.

cause of the optimization difficulty from discontinuous control point search and the limited expressiveness of the quadratic splines. The other baselines achieve reasonable performances: neural reconstructs Jell-O and Water well but struggles in all other plasticity-heavy environments due to a lack of plasticity models; gnn generally has acceptable performance in all environments. In contrast to baselines' unstable or sub-optimal performances, ours generally reaches a reconstruction loss lower than 1e-3 and ranks top-1 except for Jell-O where neural performs the best. The reason is that neural assumes no plasticity, which is precisely the case with Jell-O, and thus eases the training. We also report the initial performance of our method in ours-init to emphasize that our method learns from scratch without ad-hoc tuning for individual environments. The two oracles labeled and sys-id achieve near-perfect results in almost all environments. The only exception is labeled performing sub-optimally in Sand, which has a particularly challenging pressure-dependent plasticity law. Because labeled is supervised directly from ground-truth $\boldsymbol{P}$ and $\boldsymbol{F}^{e,\text{new}}$, we attribute this performance drop to the lack of back-propagation through time (BPTT). BPTT automatically magnifies the accumulated simulation loss and guides the neural networks to learn long-term stability.

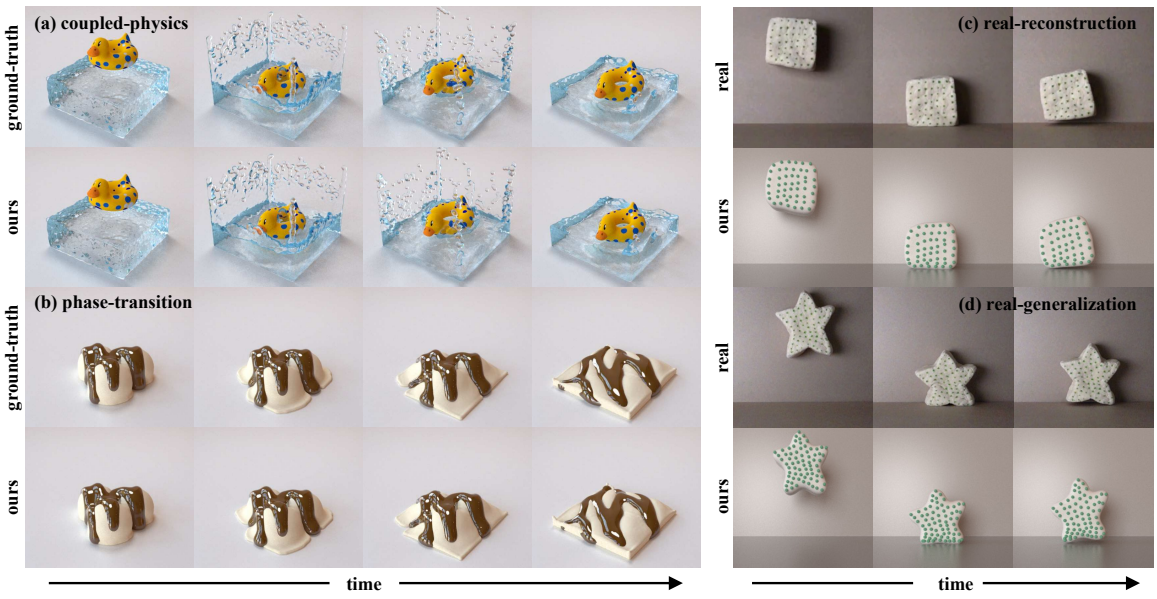


Figure 3-4: Advanced Experiments. Left: We compare our method with ground-truth results from the traditional simulation in two multi-physics environments: (a) a rubber duck falling into a swimming pool, and (b) a melting ice cream gradually changing from one material to another. Right: We train our method on the real-world data of dough dropping to learn dough's constitutive laws: (c) our method successfully reconstructs the training sequence, (d) our method generalizes well to unseen conditions, e.g., a complex geometry.

3.4.3 Generalization

After training on a *single* trajectory in Sec. 3.4.2, we directly deploy the models on four tasks with out-of-distribution conditions to evaluate the generalizability of our method: **(a)** doubled time horizon, **(b)** unseen linear and angular velocity, **(c)** challenging geometry, and **(d)** inclined plane boundary. We also used a more challenging geometry in **(d)** and increased the number of particles in **(c)** and **(d)** to $\sim$30k (30$\times$ more than training). We emphasize that single-shot generalization is a non-trivial task where the training data are only the positions of 1k particles across 1k time steps. We expand the visualization in Appendix B.6.

We select a representative subset of the experiments and report the qualitative comparison in Fig. 3-2. neural is consistent with the reconstruction performance: generalizes well on Jell-O even when the time horizon is doubled but fails on all other tasks that heavily rely on plasticity. gnn cannot generate visually plausible simulation results with different conditions than training environments. We argue that training a generalizable gnn requires

much more data than ours since it does not assume any knowledge about the underlying PDE. Indeed, Sanchez-Gonzalez et al. [154] use at least 1k motion trajectories for training on similar setups while our approach uses only one trajectory. For all tasks, ours achieves similar visual results compared to ground-truth. We also visualize ours-init to indicate the significant improvement from random initialization to a trained model.

Tab. 3.2 quantitatively compares our approach and the baseline methods. Overall, ours ranks 1st on most environments and tasks and outperforms other baselines by orders of magnitudes. Consistent with reconstruction, our method has a small enough but worse loss than neural on Jell-O. However, as shown in Fig. 3-2, this gap is so small that it only introduces a negligible visual difference between them.

3.4.4 Extreme Generalization

To study the limits of our method, we design experiments stress-testing two important factors in physical simulation: the number of particles and the collision condition. We demonsstrate the results in Fig. 3-3. On the upper row, we adopt the Water environment and apply a dragon geometry for the initial shape. We increase the number of particles in the simulation to *over 1 million* to stress-test the robustness of our method. As shown in the result, our method generates faithful prediction compared to the ground-truth results from traditional simulation even after a long horizon of time steps. We emphasize that the neural network deployed was trained on only 1k particles. On the lower row, we initialize four rubber toys made of Jell-O and throw them to each other vigorously with a large initial velocity. We depict the moment after the collision happened. Thanks to the disentanglement between the constitutive laws and the rest of the simulation, our method generalizes well to the collision condition even when it is totally unseen during training.

3.4.5 Multi-Physics Generalization

Since we train our model using a single trajectory containing only one material, it is non-trivial for it to predict the material responses in multi-physics environments. We push the boundary of this validation by constructing two challenging multi-physics environments as

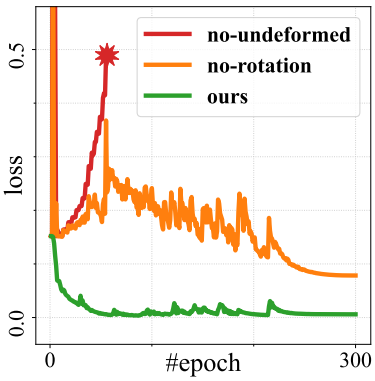
shown on the left side of Fig. 3-4. We illustrate the ground-truth motion generated using traditional simulations in the ground-truth row and compare it with the prediction of our method in the ours row. In Fig. 3-4 **(a)**, we model the rubber duck using the same material as Jell-O and drop it into a large pool of Water. Their coupling is handled by MPM. The rubber duck is modeled with 8,086 particles and the water with 94,208 particles, combined to over *100k* particles. The negligible visual difference indicates the accuracy of our method in this challenging large-scale multi-physics environment. In Fig. 3-4 **(b)**, we initialize an ice cream made of Plasticine and gradually transit the constitutive laws to Sand from bottom to top to imitate a "melting" effect. Our method remains stable and accurate even with time-varying phase transition and yields little discrepancy compared to the ground truth.

3.4.6 Real-World Experiment

Real-world materials usually present complex behaviors which are challenging and tedious to model accurately. To further evaluate our method, we choose a real-world dough as our testing material, which has rich behaviors in both elastic and plastic sides. As shown on the right side of Fig. 3-4, we record a video clip of the movement of real dough dropped from high. We dotted the dough in green to track the particle-level deformation of the material. We build the simulation directly based on the labeled particles and train our method with supervision from this real-world data. Note that the environment is *truely 3D* in order to reflect the real material property of the dough. To train 3D NCLaw with only 2D supervision, we introduce an additional prior: a regularization loss minimizing the magnitude of velocities along the depth dimension. In Fig. 3-4 **(c)**, we experimentally show that our method reconstructs the real-world trajectory of dough impacting the ground. After training, we then directly apply the trained model on an *unseen* star-shaped dough made of the same material as shown in Fig. 3-4 **(d)**. Our method captures the non-trivial elastic and plastic deformation and achieves one-shot generalization on this new complex real-world geometry. Appendix B.4 details our experimental setup and implementation.

3.4.7 Ablation Study

We then evaluate the role that rotation equivariance and undeformed state equilibrium play in training neural constitutive laws. We design two variants of our method for comparison: **no-undeformed** loosens the undeformed state equilibrium by removing the normalization of neural networks' input and adding back the bias layers. **no-rotation** loosens the rotation equivariance by directly feeding the raw deformation gradients to the neural networks. We train our method and its variants on Plasticine using our regular training configuration. As indicated by the training curve plot, our method achieves a better and faster convergence, whereas **no-rotation** is sub-optimal and **no-undeformed** blows up the simulation by introducing unstable constitutive laws.



In addition to the listed experiments, we also compare the runtime performance of our method against gnn and provide full visualization of the training dataset and generalization experiments in Appendix B.6.

3.5 Conclusions, Limitations, and Future Work

We present NCLaw as a novel hybrid NN-PDE approach toward learning generalizable PDE dynamics from motion observations. Our method enforces PDE constraints throughout the training and deployment stages while taking a data-driven approach toward constitutive modeling. Specifically, we embed NN-based constitutive laws inside a differentiable PDE-governed simulator. We then train the NN model via a loss function based on the difference between the simulator's output and the motion observation. Through a carefully designed network architecture based on physics, NCLaw guarantees common constitutive priors, such as rotational equivariance and rest state equilibrium. After training on a single motion trajectory, NCLaw achieves one-shot generalization over temporal ranges, initial/boundary conditions, complex geometries, and even multi-physics scenarios. Real-world experiments demonstrate NCLaw's ability to learn generalizable dynamics from videos.

In the near future, we aim to extend NCLaw to various physical phenomena, including heterogeneity, hysteresis, and hardening/softening. Enforcing the second law of thermody-

namics [50] in constitutive design is also an exciting direction. In terms of applications, we plan to generalize it to tasks requiring minimal sim-to-real gaps ranging from robot locomotion/manipulation to general control in complex physics [191]. We also envision extending NCLaw to other PDEs, such as the Reynold stress tensor in turbulence modeling [105]. We also plan to generalize our framework to other discretization schemes, e.g., FEM. Furthermore, our NN approach can benefit from improved interpretability and facilitating modification after training. Currently, we assume the initial condition, including the geometry, is known. Additionally, we assume that we have motion observations of the entire volumetric object (in 2D, with the entire surface). Future work may consider modeling initial condition uncertainties and working with partial observations. NCLaw can also be extended to learn from pixel-level video data without manual tracking.

At a higher level, NCLaw benefits from *both* classic PDE *and* data-driven approaches. By enforcing classic PDE priors, NCLaw achieves orders-of-magnitude higher accuracy on generalization tasks than purely NN approaches that do not enforce PDE constraints [154]. Utilizing a single expressive neural architecture, NCLaw can capture many material properties, from solids to fluids, without hand-crafted, case-by-case, expert-designed models. Consequently, we believe NCLaw will open the gates for more forthcoming hybrid NN-PDE, best-of-both-world solutions.

Chapter 4

LLM and Simulation as Bilevel Optimizers: A New Paradigm to Advance Physical Scientific Discovery

4.1 Introduction

In physical science, spanning physics, chemistry, pharmacology, etc., various research streams aim to automate and speed up scientific discovery [189]. Each stream innovates within its field, creating methods tailored to its specific challenges and nuances. However, this approach often misses a universally applicable philosophy [139, 41], which can be pivotal to democratizing access to advanced research tools, standardizing scientific practices, and enhancing efficiency across disciplines. Our goal aims to transcend specific domains, offering a unified approach to physical science.

As an inspiration, we observe how human scientists conduct scientific discovery experiments and conclude a few key experiences: (i) iteratively propose a hypothesis and make observations from experimentation to correct theories [139], (ii) divide the solutions into discrete components, such as physics equations or molecule structures, and continuous components, such as parameters for physics and molecule properties [189], (iii) exploit the existing knowledge while occasionally explore novel ideas aggressively in pursuits of break-

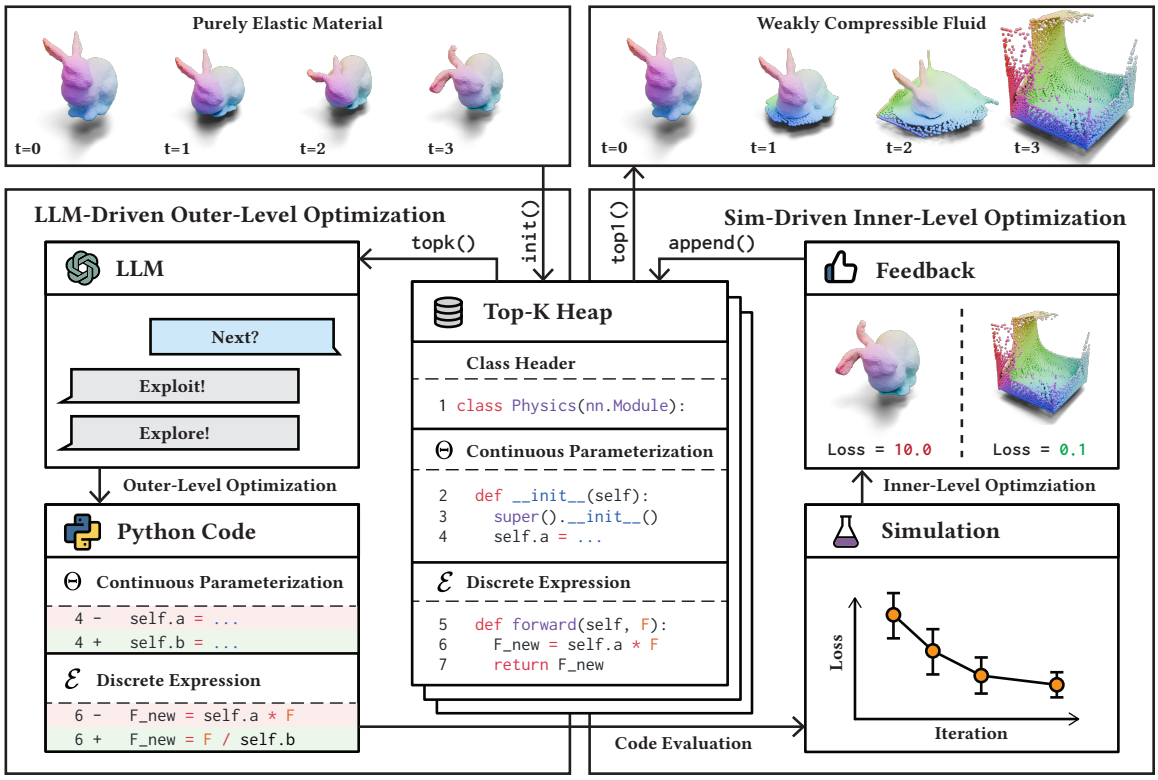


Figure 4-1: The overall pipeline of Scientific Generative Agent (SGA). Taking the constitutive law searching problem as an example, the input is an initial guess (a purely elastic material), and the output is another constitutive law optimized towards the ground-truth (weakly compressible fluid). The initial guess first initialize a top-K heap for storing the solutions. In the outer-level optimization, an LLM takes in top-K previously proposed solutions and generates a better one upon them with modified continuous parameterization Θ and discrete expression $\mathcal{E}$. In the inner-level optimization, a gradient-based optimization solves for optimal Θ via simulation and appends these optimized solutions in the heap. After a few iterations of bilevel optimization, the heap returns the top-1 solutions as the final solution.

through [195], (iv) follow a generic, universal principle for all types of physical scientific discovery yet with specific nuance of each discipline [152].

Standing out as generalist tools with an extensive repository of knowledge [2], large language models (LLMs) have recently risen to prominence in scientific discovery for their expansive knowledge bases, advanced reasoning capabilities, and human-friendly natural language interface. One line of research focuses on fine-tuning LLMs with domain-specific data to align natural language with scientific information, such as chemical [27] or drug [110] structures; however, these methods are domain-bound and demand extensive data for

broader application. Another research direction seeks to leverage the innate capabilities of pre-trained LLMs, augmented by external resources like the internet, programming, or documentation. LLMs serve as optimizers or agents [69] for mathematical problem-solving [151], conducting chemical experiments [15], and advancing molecular [96] and drug discovery [157]. Nevertheless, these approaches are confined to the computational capability of LLMs, a crucial factor in physical science for tasks like calculating numerical results based on physics law hypotheses to predict natural phenomena. To address this limitation, we propose to augment LLMs with physical simulation, hereby merging the knowledge-driven, abstract reasoning abilities of LLMs with the computational structures and accuracy of simulations.

To this end, inspired by the overarching philosophy of human scientists, we introduce **Scientific Generative Agent (SGA)**, a bilevel optimization approach wherein the outer-level engages LLMs as knowledgeable and versatile thinkers for generating and revising scientific hypothesis, while the inner-level involves simulations as experimental platforms for providing observational feedback. First, we employ LLMs to generate hypotheses, which then guide the execution of simulations. These simulations, in turn, yield observational feedback that helps refine and improve the proposed hypotheses. Secondly, we introduce a bilevel optimization framework: one level performs search-based optimization on discrete, symbolic variables like physics laws or molecule structures via LLMs; the other level performs gradient-based optimization via differentiable simulation for continuous parameters like material stiffness or molecule coordinates. Thirdly, we devise an exploit-and-explore strategy for the hypothesis proposal by adjusting LLM's generation temperature. Lastly, we demonstrate our pipeline is generally applicable across scientific disciplines, with only minimal modification such as altering the prompts.

For the empirical study, we focus on (i) *molecular design* that aims to discover molecular structure and atoms' coordinates based on its conformation and quantum mechanical properties and (ii) *constitutive law discovery* that aims to discover material constitutive equations and its corresponding mechanical properties directly from a recorded motion trajectory. To provide a concrete example, let's assume that we initially have simply the code for a purely linear material. We then task our model to uncover a more complex representation by

optimizing its code to fit a highly non-linear trajectory. In this task, our method capitalizes on the strengths of bilevel optimization: the outer-level utilizes LLMs to identify the correct symbolic material constitutive equations and formulates a proposition for potentially beneficial continuous parameterization (e.g., Young's modulus and Poisson's ratio); and the inner-level refines the proposed material parameters and provides informative feedback using differentiable simulation. Generally, our method can discover the desired molecules and constitutive laws, outperforming other LLM-based baselines; more interestingly, it can propose well-performing solutions that are beyond human expectation yet sensible under analysis by domain experts. Overall, our contributions are concluded as:

- We present a generic framework for physical scientific discovery that combines LLMs with physical simulations.
- We propose a bilevel optimization with LLMs for discrete-space search-based optimization and differentiable simulations for continuous-space gradient-based optimization.
- We conduct extensive experiments to demonstrate the effectiveness and generality of the proposed framework in physics law discovery and molecular design; moreover, we showcase novel molecules or constitutive laws, while unexpected from a conventional perspective, are deemed reasonable upon examination by domain experts.

4.2 Scientific Generative Agent

SGA is a bilevel optimization framework where the upper level features LLMs as proposers of scientific solutions, and the lower level utilizes simulations as experimental platforms for validation. In section 4.2.1, we describe a formal definition of the bilevel optimization, followed by section 4.2.2 for outer optimization and section 4.2.3 for inner optimization.

4.2.1 Bilevel Optimization Pipeline

We formally describe the pipeline of our method, including the input/output of the system and the underlying submodules, and the overall optimization formulation. Suppose we are

given a metric to evaluate a physical phenomenon y (e.g., a configuration of deformation) for a scientific problem $\mathcal{L}(y)$ (e.g., reconstruction of a mechanistic behavior). First, we describe the simulation (as an experimental platform) as,

$$y, z = \Phi(\theta; \mathcal{E}), \tag{4.1}$$

where Φ is a simulator that takes in scientific expression $\mathcal{E}$ (e.g., constitutive equations) and continuous components θ (e.g., material parameters) as inputs and gives simulated physical phenomenon y and additional observational feedback z (e.g., particles' trajectories) as outputs. Next, the LLM is prompted to act as a thinker to propose expressions $\mathcal{E}$ based on past experimental results from simulation,

$$\mathcal{E}, \Theta = \text{LLM}\left(\left\{\mathcal{L}(y_k), z_k, o_k, \mathcal{E}_k, \Theta_k\right\}_{k\in[K]}; \mathcal{P}\right), \tag{4.2}$$

where the set $[K]$ summarizes the pointers to the past simulation results containing an evaluation of the scientific problem $\mathcal{L}(y_k)$, other physical feedback (z_k, o_k), and past proposals $(\mathcal{E}_k, \Theta_k)$; o_k summarizes the intermediate results of the inner optimization (later detailed in section 4.2.3); Θ determines the continuous parameterization for the decision variables of the inner optimization (e.g., which variables to be optimized within a proposed equation); $\mathcal{P}$ is prompt. With these, we define the bilevel optimization problem as,

$$\min_{\mathcal{E},\Theta} \mathcal{L}\left(y\left(\mathcal{E}, \Theta, \hat{\theta}; \Phi\right)\right) \tag{4.3a}$$

$$\text{s.t. } G(\mathcal{E}, \Theta; \Phi) \leq 0 \tag{4.3b}$$

$$\hat{\theta} \in \arg\min_{\theta\in\Theta} \mathcal{L}(y(\theta; \Phi, \mathcal{E})), \tag{4.3c}$$

where $G(\cdot) \leq 0$ refers to the validity of the simulation (i.e., whether an expression $\mathcal{E}$ is simulatable). The outer optimization searches for (i) an expression $\mathcal{E}$ that defines what experiments to be conducted $\Phi(\cdot; \mathcal{E})$ and (ii) continuous parametrization Θ that defines the search space of the inner continuous optimization $\min_{\theta\in\Theta}$. With the dependencies on the outer-level variables $(\mathcal{E}, \Theta)$, the inner optimization searches for the optimal continuous

Algorithm 3 Scientific Generative Agent

Input: Discrete expression and continuous param $(\mathcal{E}, \theta \in \Theta)$,
Num of exploiting M_l, Num of exploring M_h,
Exploiting temperature T_l, Exploring temperature T_h

```
# Store ranked (solution,param) by heap
H ← heap()
# Continuous optimization
θ̂ ← optim(E,θ;Φ)
H.append((E,θ̂))
for i = 1,...,N do
  # Generate M_l solutions from LLM
  (E,Θ)[:M_l] ← LLM(H.topk(K),T_l)
  # Generate M_h solutions from LLM
  (E,Θ)[M_l:M_l+M_h] ← LLM(H.topk(K),T_h)
  for m = 1,...,M_l + M_h do
    # Continuous optimization
    θ̂ ← optim(E,θ ∈ Θ;Φ)
    H.append((E,θ̂))
  end for
end for
```

Output: `H.topk(1)` `# Return the best`

parameters $\hat{\theta}$ given the proposed expression via differentiable simulation.

4.2.2 LLM-Driven Outer-Level Search

We dive deeper into how we use LLMs (equation 4.2) and their interaction with the simulation for outer-level search (equation 4.3).

LLM-driven Optimization LLMs have shown to be effective sequential decision makers for generic optimization, providing proper guidance via prompting and sufficiently informative contexts [201, 151]. We craft prompts to direct LLMs in a structured manner, enabling them to (i) perform *analysis* on past experimental results from the simulation, e.g., the deviatoric parts of stress tensor are likely correct based on the loss curve; (ii) devise a high-level *plan* on how to formulate a hypothesis or improve upon previous experiments, e.g., ensure numerical stability with the usage of the determinant of deformation gradient; (iii) suggest a *solution* that can be executed as experiments via simulation for hypothesis testing; e.g., a code snippet describing a constitutive equation. For equation 4.3a, inspired by [117],

we adopt an evolutionary search that generates multiple offspring $\{\mathcal{E}_m, \Theta_m\}_{m\in[M]}$ (M is offspring size) in each iteration and retain the best selection. Distinctively, our approach (algorithm 3) involves selecting several high-performing candidates rather than the best only, which (i) enhances the feasibility of hypotheses in simulation (equation 4.3b) and (ii) facilitates evolutionary crossover, with LLMs generating new hypotheses from various past experiments ("breeds") for better exploration, akin to the findings in [151].

Interfacing with Simulation The primary challenge in integrating LLMs with simulation lies in devising a protocol that enables efficient, structured, yet adaptable communication between the two modules. We observe that physical scientific solutions are often represented as mathematical expressions or structured entities. Hereby, from LLMs to simulation, we consider two settings: equation searching and entity searching, both unified as the abstraction $(\mathcal{E}, \Theta)$ in equation 4.2. In *equation searching*, LLMs are allowed to propose equations $\mathcal{E}$ along with the search space of the inner-level continuous optimization Θ; for practitioners, an example using PyTorch can be Θ as `__init__` that defines continuous parameters via `nn.Parameter` and $\mathcal{E}$ as `forward` that defines computation of equations (see figure 4-1). In *entity searching*, LLMs propose descriptions of structures $\mathcal{E}$ (e.g., how atoms are connected to form a molecule) with Θ simply reduced to constant (e.g., every atom has its 3D coordinates to be optimized) and omitted from the optimization equation 4.3a as decision variables. On the other hand, from simulation to LLMs, we leverage domain experts' knowledge to craft functions for extracting compact, relevant information z as observational feedback; this process is akin to an experienced scientist offering guidance to a junior colleague on how to document experimental findings effectively. For instance, human experts often monitor the movements of specific body regions to derive constitutive laws. Therefore, to aid in this process, we include a function in the simulation that records the particle trajectories. Lastly, the subsequent section section 4.2.3 will provide an in-depth explanation of the inner optimization results denoted as o. These results serve as feedback from the simulation to the LLMs.

Exploitation and Exploration Inspired by human scientists achieving breakthroughs by skillfully balancing careful progression with bold exploration, we devise an exploit-and-explore strategy by tuning the LLMs' decoding temperature [201]. When generating offspring $\{\mathcal{E}_m, \Theta_m\}_{m\in[M]}$ in equation 4.3a, we divide them into two groups: one ($m \in \mathcal{M}_{\text{exploit}}$) consists of cautious followers that keep the "gradient" and conservatively trails previous solutions, while the other ($m \in \mathcal{M}_{\text{explore}}$) comprises daring adventurers that take risks and suggest unique solutions. Empirically, we observed that (i) $\mathcal{M}_{\text{exploit}}$ often contains repetitive solutions from previous iterations, and (ii) $\mathcal{M}_{\text{explore}}$ tends to yield solutions too random to be informative for guiding optimization, or invalid (i.e., violating equation 4.3b), thus providing little feedback signal. As a rule of thumb, we have found that a 1:3 ratio between $\mathcal{M}_{\text{exploit}}$ and $\mathcal{M}_{\text{explore}}$ is effective.

4.2.3 Differentiable Inner-Level Optimization

Under the search space Θ and expression for simulation $\mathcal{E}$ from the outer level, inner optimization (equation 4.3c) involves a gradient-based optimization that solves for optimal continuous parameters $\hat{\theta} \in \Theta$ via differentiable simulation (equation 4.1). Essentially, the domain-specific knowledge is distilled via gradients $\nabla_\theta \Phi(\theta; \mathcal{E})$ from the simulation to the intermediate optimization results o (like loss curve). The $(\hat{y}, o)$ are then fed back to LLMs for revising solutions. Note that o may involve the loss curve toward the target metric $\mathcal{L}$ and other auxiliary recordings throughout optimization, carrying information of how to improve solutions in various aspects; for example, with $\mathcal{L}$ as displacement of position, o may include velocities across the inner optimization iterations.

4.3 Experiments

4.3.1 Problem Definitions

Constitutive Law Discovery Identifying the constitutive law from motion observations stands as one of the most difficult challenges in fields such as physics, material science, and mechanical engineering. Here we follow the recent advances in physical simulation

Table 4.1: Benchmark. We compare our method against 4 baselines and 2 variations of our method, while also noting the difference in architecture or hyper-parameters. We use column #Iter. as the number of iterations, #Hist. as the K value for the top-k retrieval in the historical optimization steps, $\frac{\#\text{Exploit}}{\#\text{Explore}}$ as the number of offspring for exploitation versus exploration, Bilevel as if bilevel optimization is enabled. Our experiments encompass 8 different tasks, which are divided into constitutive law search (a-d) and molecule design (e-h). A lower loss value is preferable across all tasks. The best method with the lowest loss is highlighted in bold text.

Method	#Iter.	#Hist.	$\frac{\#\text{Exploit}}{\#\text{Explore}}$	Bilevel	Constitutive Law Search				Molecule Design			
					(a) ↓	(b) ↓	(c) ↓	(d) ↓	(e) ↓	(f) ↓	(g) ↓	(h) ↓
CoT	1	5	N/A	✗	298.5	1462.3	150.0	384.1	3.0	32.1	18.6	6.0
FunSearch	20	2	0 / 4	✗	210.3	872.2	82.8	139.5	1.1	7.1	8.3	1.1
Eureka	5	1	0 / 16	✗	128.0	531.0	101.7	150.1	4.3	9.8	3.3	9.7e-1
OPRO	5	5	0 / 16	✗	136.2	508.3	99.2	128.8	2.4	9.4	3.1	1.3
Ours (no bilevel)	5	5	4 / 12	✗	90.2	517.0	83.6	68.4	8.6e-1	9.1	1.8	1.4
Ours (no exploit)	5	5	0 / 16	✓	3.0e-3	3.9e-1	6.6e-2	**1.4e-12**	4.0e-4	1.5e-1	6.1e-1	**2.8e-5**
Ours	5	5	4 / 12	✓	**5.2e-5**	**2.1e-1**	**6.0e-2**	**1.4e-12**	**1.3e-4**	**1.1e-1**	**5.4e-1**	3.6e-5

and formulate the constitutive law discovery task as an optimization problem [115] using differentiable Material Point Method (MPM) simulators [164, 76]. Note that our method is not specifically tailored to MPM simulators and applies to any physical simulation. The objective of this task is to identify both the discrete expression and continuous parameters in a constitutive law, specifically the symbolic material models $\varphi(\cdot)$ and their corresponding material parameters θ, from a ground-truth trajectory of particle positions $\hat{X}_{t\in[1,\ldots,T]}$ where T denotes the number of steps. In this problem, we consider two types of constitutive laws, $\varphi_E(\cdot;\theta_E)$ and $\varphi_P(\cdot;\theta_P)$, for modeling elastic and plastic materials respectively, and they are formally defined as:

$$\varphi_E(\mathbf{F};\theta_E) \mapsto \boldsymbol{\tau} \tag{4.4a}$$

$$\varphi_P(\mathbf{F};\theta_P) \mapsto \mathbf{F}^{\text{corrected}}, \tag{4.4b}$$

where $\mathbf{F} \in \mathbb{R}^{3\times3}$ is the deformation gradient, $\boldsymbol{\tau} \in \mathbb{R}^{3\times3}$ is the Kirchhoff stress tensor, $\mathbf{F}^{\text{corrected}} \in \mathbb{R}^{3\times3}$ is the deformation gradient after plastic return-mapping correction, and θ_E and θ_P are the continuous material parameters for elastic and plastic constitutive laws respectively. Given a specific constitutive law, we input it to the differentiable simulation

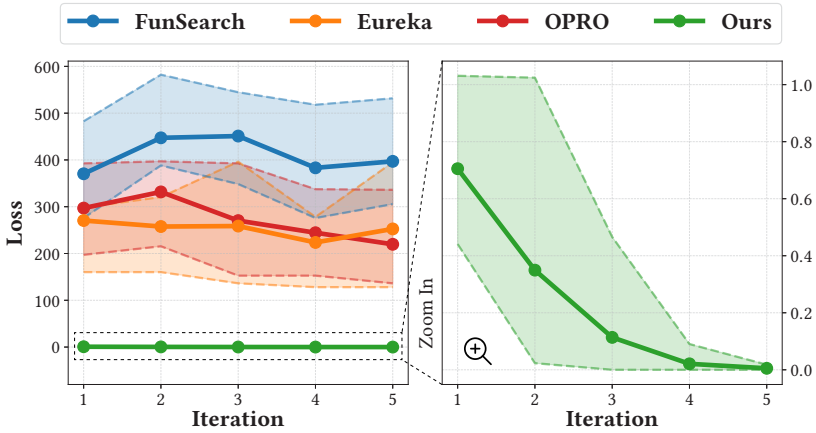


Figure 4-2: Loss trends comparison. Loss of the best solution averaged across seeds at different iterations of LLM-driven optimization, where the shading shows the min/max value.

and yields a particle position trajectory:

$$X_{t\in[1,\ldots,T]} = \mathrm{sim}\left(\varphi\left(\cdot;\theta\right)\right), \tag{4.5}$$

and we optimize the constitutive law by fitting the output trajectory to the ground truth $\hat{X}_{t\in[1,\ldots,T]}$.

Molecule Design In this study, we focus on a prevalent task in molecule design: discovering molecules with specific quantum mechanical properties. Our objective is to determine the optimal molecular structure and its 3D conformation to match a predefined target quantum mechanical property. The design process involves both the discrete expression – the molecular structure represented by SMILES strings [194], and the continuous parameters – the 3D coordinates of each atom in the molecule. The methodology comprises two loops: In the outer loop, the LLM generates the initial molecular structure as a SMILES string, along with a preliminary guess for the 3D atom coordinates. The inner loop involves simultaneous optimization of both the molecule's 3D conformation and quantum mechanical properties, both determined by 3D atom positions.

For the generation of 3D conformations, we utilize the ETKGD algorithm [150] followed by optimization using the Merck Molecular Force Field (MMFF) [56], both implemented within the RDKit [93]. To get the quantum mechanical property values, we employ Uni-

Mol [212], a pre-trained transformer-based large model, which has been fine-tuned on the QM9 dataset [148].

4.3.2 Experiment Setup

Task Design We design a diverse set of challenging tasks for evaluation. For constitutive law discovery, we propose 4 tasks including: **(a)** fitting the non-linear elastic material starting from a linear elastic material, **(b)** fitting the von Mises plastic material starting from a purely elastic material, **(c)** fitting the granular material starting from a purely elastic material, and **(d)** fitting the weakly compressible fluid starting from a purely elastic material. For molecular design task, we consider 4 popular tasks, centering on 3 commonly evaluated quantum mechanical properties [40, 212], each set to different target values: **(e)** HOMO (Highest Occupied Molecular Orbital) set to 0, **(f)** LUMO (Lowest Unoccupied Molecular Orbital) set to 0, **(g)** the HOMO-LUMO energy gap set to 0, and **(h)** the HOMO-LUMO energy gap set to -2. All these values are normalized on all data in QM9 dataset.

Implementation Details We run all our experiments 5 times with different random seeds following previous practices [117]. Due to the complexity of the task, we provide a simple bootstrapping example of a valid design to ensure the success rate. We use warp [118] for the differentiable MPM simulation, and we develop our inner-level optimization upon PyTorch [134]. In all our experiments, we use mean square error as the criteria and Adam optimizer [84]. We choose `gpt-4-turbo-preview` as the backbone model for LLM and tentatively set the exploiting temperature $T_l = 0.5$ and exploring temperature $T_h = 1.0$.

4.3.3 Physical Scientific Discovery

We consider 6 strong baselines for evaluation: (i) **Chain-of-Thoughts (CoT)** prompting [193] solves the problem by looking at step-by-step solutions from examples. We provide 5 examples with an explanation to CoT as the initial solution. (ii) **FunSearch** [151] utilizes evolutionary strategy to avoid local optimum. We adopt the given hyperparameters from the original implementation with 2 optimization histories and 4 explorers. We set

Table 4.2: Comparison with symbolic regression. We compare our method against 5 most performant methods in SRBench and 3 pre-trained symbolic regression methods. Sym. denotes whether the result is symbolic or not.

Method	R2 ↑	MSE ↓	MAE ↓	Sym.
FFX	0.9824	4.5e+5	3.7e+2	✓
MLP	0.9876	3.2e+5	3.4e+2	✗
FEAT	0.9964	9.2e+4	1.7e+2	✓
DSO	0.9968	8.2e+4	9.2e+1	✓
Operon	0.9988	2.8e+4	9.8e+1	✓
SymbolicGPT	0.5233	6.9e+6	1.7e+3	✓
NeSymReS	N/A to >3 variables			✓
T-JSL	N/A to >2 variables			✓
Ours	**0.9990**	**1.7e+4**	**8.6e+1**	✓

Table 4.3: Comparison with population-based molecule design. We compare our method against a traditional population-based molecule design method GhemGE and report the results of molecule design tasks (e-h).

Method	(e) ↓	(f) ↓	(g) ↓	(h) ↓
GhemGE	4.8e-3	1.8	1.5	9.8e-5
Ours	**1.3e-4**	**1.1e-1**	**5.4e-1**	**3.6e-5**

the number of iterations to 20, yielding the same number of solutions evaluated, for a fair comparison to other methods. (iii) **Eureka** [117] generates multiple solutions in each iteration to improve the success rate of the generated code. We keep the hyperparameters from the original implementation. (iv) **Optimization by PROmpting (OPRO)** [201] highlights the advantages of involving a sorted optimization trajectory. We set the hyperparameters to be equal to **Eureka** except for the number of historical optimization steps. In all these works (i-iv), we notice the temperatures for LLM inference are all 1.0, which is equal to the exploring temperature in our method, so we denote them with 0 exploiter. We also consider 2 variants of our method: (v) **Ours (no bilevel)** removes the bilevel optimization by only searching with LLM. (vi) **Ours (no exploit)** removes the exploitation by setting the temperature to 1.0 all the time.

We present our experiments against the 8 designed tasks and show the results in Table 4.1. Compared to baselines (i-iv), our method is significantly better by a number of

Table 4.4: Experiment in imaginary constitutive law. We construct an imaginary constitutive law to keep LLM from cheating by memorization and report the results of our method and baselines.

Method	**FunSearch**	**Eureka**	**OPRO**	**Ours**
Loss	105.0	89.1	98.0	**1.3e-3**

magnitudes. When the bilevel optimization is removed from our method, the performance drops dramatically, but still statistically better than baselines (i-iv), indicating the choice of hyperparameters and the integration of exploitation is helpful for the task. When we remove the exploitation but restore the bilevel optimization, we notice the performance grows back. It has comparable performance compared to our method in **(d)** or even better results in **(h)**. However, in some tasks, especially hard ones (e.g., **(b)** and **(f)**) that we care more in reality, the performance gap is over 50%, indicating the effectiveness of our exploit-and-explore strategy. We also present the loss trend in task **(a)** in Figure 4-2, our method outstands with a much lower loss and a converging trend.

We also compare our method with traditional methods in each specific area to demonstrate the generalizability of our method. First, we reformulate our constitutive law search task **(a)** into a symbolic regression task by (i) capture the ground-truth output (the stress tensors) as the supervision, and (ii) separate the 9 output dimension into 9 independent problems and ensemble them for evaluation. Note that these modifications dramatically simplified the original task: we removed back-propagation through time (BPTT) and directly discover the constitutive law without surrogate loss. We evaluate 14 traditional baselines in SRBench [91] and 3 data-driven pre-trained baselines. We select the top few baselines in Table 4.2 and show the rest in the Appendix C.3.1. As shown the table, our method topped on this task even with a much more challenging setting. Also, since our method depends on the in-context learning ability of LLMs, it has little constraint in the number of variables than the data-driven pre-trained baselines. For moledule design tasks, we also compare our method with GhemGE [205], which employs a population-based molecule design algorithm. As shown in Table 4.3, our method presents a much lower loss, demonstrating the general effectiveness of our method.

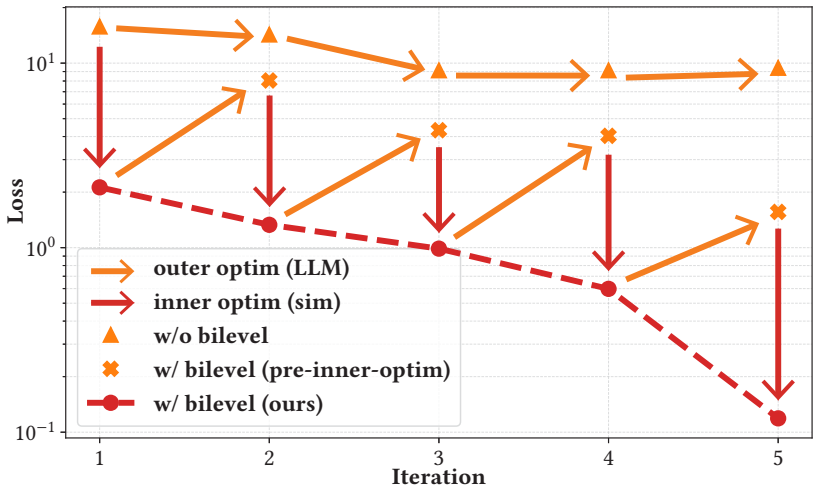


Figure 4-3: Ablation on bilevel optimization. We denote the optimization trajectory with and without out bilevel optimization with red dot and orange triangle respectively. We visualize the intermediate step of our method before the inner-level optimization using orange cross. We also highlight the outer LLM optimization and inner simulation optimization using orange and red arrows.

4.3.4 Ablation Study

Generalization or Memorization In order to figure out if the improvement introduced by our method is merely because the LLM saw the solutions in its training phase, we design an experiment ablating it by making it invent an imaginary constitutive law that does not exist on the earth. We mix the constitutive law of von Mises plasticity, granular material, and weakly compressible fluid by 50%, 30%, and 20%, so that the new constitutive law represents an imaginary material whose behavior is extremely complex. We repeat our experiment setup as in Figure 1. We compare our method against the baselines and report the performances in Table 4.4. As shown in the table, our method can still discover the constitutive law with a low quantitative loss. From our observation, there is *very little visual difference* between the ground-truth material and the optimized constitutive law. We show the discovered constitutive law in Appendix C.4.5.

Bilevel Optimization is the Key Here we evaluate the importance of bilevel optimization in Figure 4-3 using the task **(h)**. Comparing the blue triangle curve and the red dot curve, which represent the LLM-driven outer-level optimization and the simulation-driven inner-level optimization, it is easy to conclude that the loss performance with bilevel optimization

is better. Nevertheless, we are also interested in how bilevel optimization works inside each optimization step and how much LLMs and simulations help respectively. As shown as a zigzag curve, we found that LLMs and simulations help each other over all optimization steps: the next proposal from LLMs will be better with simulation-optimized results, and vice versa. We argue that LLMs and simulations have different expertise: LLMs are generalist scientists who have cross-discipline knowledge, while simulations are domain experts who have specialized knowledge.

LLM Backbone In addition to GPT-4 [130], we repeat the experiments in Table 4.1 using 3 additional LLM backbones: (i) GPT-3.5 [133], (ii) Claude-3-Sonnet [5], and (iii) Mixtral-8x7B [74], and report the rank of them in Figure 4-4. Indicated by the largest area, GPT-4, as our choice, statistically outperforms the other methods. Interestingly, we found Claude-3-Sonnet is the second top method on most of constitutive law search task, while Mixtral-8x7B even tops on 2 molecule design tasks. As a result, our workflow also works for other LLMs, however, our suggestion for practitioners is to try GPT-4 as the first choice but also consider open-source model (e.g., Mixtral-8x7B) for budget or customizability.

Exploitation v.s. Exploration We visualize the statistics of the simulation execution status in Figure 4-5 (a) using the task **(b)**, which is one of the most challenging tasks in our experiments. When the exploitation is removed, the error rate dramatically increases, as shown by a decrease in green bars. It leads to a degeneration in the performance of the methods with exploitation as shown in Figure 4-5 (b). However, even though the success rate remains high, when exploration is removed, the optimization result is still worse than keeping them both. We argue that exploration is significant when the optimization problem is challenging, especially in our case, where the search space is highly non-linear and unstructured and resulting in numerous local optimum.

4.3.5 Case Study

Constitutive Law Search We provide a trimmed snippet of our searched constitutive law in Figure 4-6 (a) for task **(a)** where a highly non-linear material is provided as the

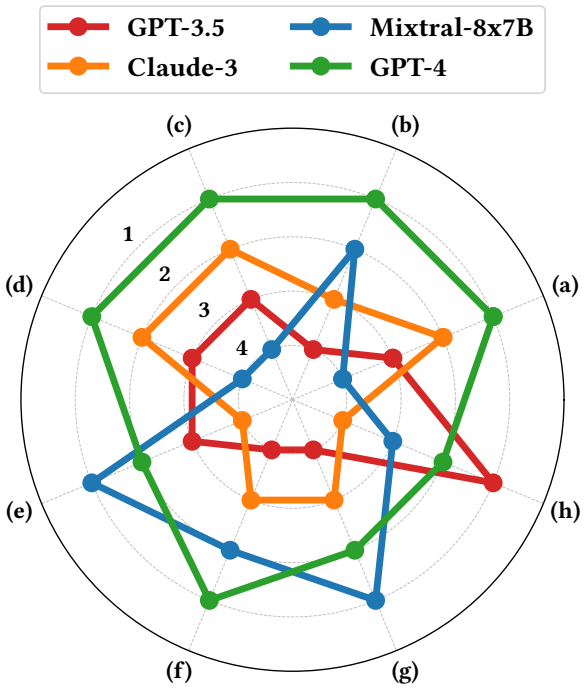


Figure 4-4: Ablation on the backbone LLM. We compare the performances of 4 selected backbone LLMs and report the rank of them. A outer curve indicates a better performance.

trajectory to fit. We reformat the code slightly to fit into the text, where the complete example can be found in the Appendix. Starting from a linear material, our method is able to automatically generate the constitutive law with a quadratic deviatoric term. Note that our method also provides a concrete implementation of `__init__` function that defines the continuous parameters in the computational graph for later inner-level optimization.

Molecule Design When comparing the two molecules with respect to their HOMO-LUMO energy gap based on optimized results from the LLM as shown in Figure 4-6 (b), we observe distinct characteristics in each: (i) **Molecule A** (gap-0) includes sulfur and chlorine atoms attached to a ring, coupled with a trifluoromethyl group, introducing electron-withdrawing effects, and (ii) **Molecule B** (gap-2) includes oxygen (notably in ethers) and sulfur within the ring structures introducing localized non-bonding electron pairs. Furthermore, the overall structure of Molecule B is more complex than that of Molecule A, containing multiple rings. An intriguing aspect of Molecule B, which might initially defy expectations, is the presence of a single fluorine atom. The high electronegativity of fluorine typically leads to electron density withdrawal, influencing the gap value.

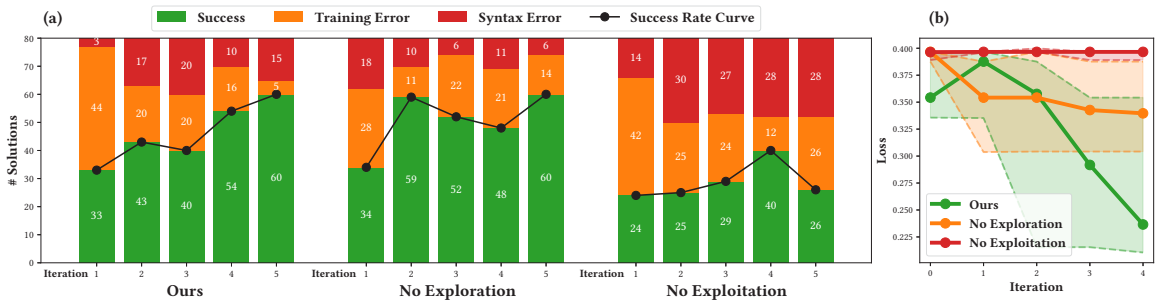


Figure 4-5: Ablation on exploration-exploitation. (a) Histogram of solutions that are valid for simulation across iterations. (b) Loss of the best solution averaged across seeds at different iterations, where the shading indicates the min/max values.

However, due to the complexity of Molecule B's structure, the impact of the fluorine atom is somewhat localized, thereby not significantly altering the gap value.

4.4 Related Work

4.4.1 Automated Scientific Discovery

Automated scientific discovery, enhanced by machine learning methods, serves as a powerful accelerator for research, enabling scientists to generate hypotheses, design experiments, interpret vast datasets, and unearth insights that may elude traditional scientific methodologies [2, 89, 189]. This multifaceted process unfolds through two synergistically linked stages: hypothesis formation and the collection and analysis of experimental data. The integration of automated systems not only augments the scientific inquiry process but also streamlines the discovery pipeline, from conceptualization to empirical validation. This paper places a particular emphasis on, but is not limited to, constitutive law discovery and molecular design. These areas exemplify the profound impact of automation in unraveling complex material behaviors and in the innovative design of molecules with tailored properties. Automatic identification of constitutive material models has been a long-standing problem and recent works utilizes differentiable simulation [37, 115, 113] to address it as a system identification problem. Leveraging machine learning and artificial intelligence, researchers are able to predict molecular behavior, optimize chemical structures for specific functions, and thus, rapidly accelerate the development of new drugs, materials, and

(a)

```python
class Physics(nn.Module):
  def __init__(self, youngs_modulus_log: float = 13.03,
                     poissons_ratio_sigmoid: float = -1.99):
    super().__init__()
    self.youngs_modulus_log = nn.Parameter(
      torch.tensor(youngs_modulus_log))  # Log of Young's modulus
    self.poissons_ratio_sigmoid = nn.Parameter(
      torch.tensor(poissons_ratio_sigmoid))  # Sigmoid of Poisson's ratio
  def forward(self, F: torch.Tensor) -> torch.Tensor:
    youngs_modulus = self.youngs_modulus_log.exp()
    poissons_ratio = torch.sigmoid(self.poissons_ratio_sigmoid) * 0.49
    mu = youngs_modulus / (2 * (1 + poissons_ratio))  # Shear modulus
    lam = youngs_modulus * poissons_ratio / (
      (1 + poissons_ratio) * (1 - 2 * poissons_ratio))
    # Deformation gradient determinant J
    J = F.det().view(-1, 1, 1)
    F_invT = F.inverse().transpose(1, 2)
    # Volumetric part
    P_vol = lam * (J - 1) * F_invT
    # Deviatoric part
    P_dev = mu * (F - (1 / J) * F_invT)
    # Compute Kirchhoff stress tensor
    kirchhoff_stress = P_vol + P_dev @ F.transpose(1, 2)
    return kirchhoff_stress
```

(b)

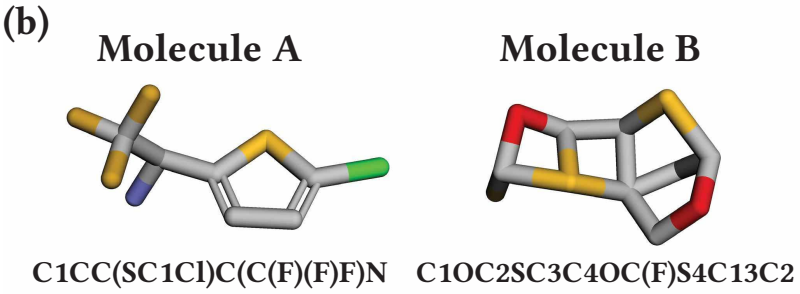


Figure 4-6: Case Study. (a) We give a concrete example of the searched constitutive law. (b) We provide 2 novel molecules optimized for different objectives with their SMILES stings.

chemicals [77, 213, 156].

4.4.2 Large Language Models and Agents

The advancement of Large Language Models (LLMs) such as ChatGPT and GPT-4 has sparked considerable interest in their potential as autonomous agents [18, 129, 130]. Recent developments have shown that LLMs can be enhanced to solve complex problems by creating and utilizing their own tools, as demonstrated in the LATM framework [165], and by acting as optimizers in the absence of gradients, as seen in the OPRO methodology [201]. These approaches signify a shift towards more independent and versatile LLM-based agents capable of generating solutions through self-crafted tools and optimization

techniques [20, 203, 202], showcasing their evolving problem-solving capabilities. In the realm of scientific discovery, LLMs have begun to make significant contributions, particularly in mathematics and computational problems. The FunSearch method [151] pairs LLMs with evaluators to exceed known results in extremal combinatorics and online bin packing, illustrating LLMs' ability to discover new solutions to established problems. Similarly, AlphaGeometry's success [173] in solving olympiad-level geometry problems without human demonstrations highlights the potential of LLMs in automating complex reasoning tasks. These examples underline the transformative impact of LLMs in pushing the boundaries of scientific inquiry and automated reasoning.

4.4.3 Bilevel Optimization

Bilevel optimization involves a hierarchical structure with two levels of optimization problems, where the solution to the upper-level problem is contingent upon the outcome of the lower-level problem [28]. Bilevel optimization problems are inherently more complex than their single-level counterparts due to the nested nature of the optimization tasks and the intricate interdependencies between them. Recent advancements have focused on developing efficient algorithms, including evolutionary algorithms [160], gradient-based approaches [109], and approximation techniques [159], to tackle the computational challenges presented by the non-convex and non-differentiable characteristics of many bilevel problems. Among a wide span of application domains of bilevel optimization, neural architecture search (NAS) [107, 12, 19, 200] is prominent and close to the problem setting in this paper: the upper level optimizes the discrete neural network architecture while the lower level optimizes the continuous weights of the neural network. However, typical NAS methods require a predefined search space, constraining the exploration of discrete network architectures to manually specified boundaries. Our framework distinguishes itself by employing LLM encoded with general knowledge and gets rid of the limitations imposed by manual design constraints.

4.5 Conclusion

We consider a few limitations and future directions. (i) Although we prompt the LLM to generate pseudo-code plans and comments, it is generally hard to ensure the interpretability of LLM-generated solutions. (ii) Since the LLM-generated codes are executed directly without any filtering in our application, there exists potential AI safety risk that hazards the operating system. (iii) Our method only utilizes the internal knowledge of LLMs as the prior, where in reality people design manual constraints and rule to regularize and improve the optimization [176]. We leave these domain-specific applications and human feedback-based regularization methods as our future work. (iv) The performance our method highly depends on the differentiablity of the generated code. However, Zero-order optimizers [59] should also shine since the number of continuous parameters is relatively limited. (v) LLM inference requires large computational resources and thus increases expense. For example, it spends around $10 for our method to complete one task using GPT-4, which will be increasingly inacceptable when the number of iteration grows. (vi) Due to the reuse of previously generated solutions in our proposed top-k heap, the KV cache in LLM will be highly similar between neighbor iterations. It opens a gate for recent KV cache optimization methods [210] to speedup our method by KV cache reusing.

In conclution, we present Scientific Generative Agent, a bi-level optimization framework: LLMs serve as knowledgeable and adaptable thinkers, formulating scientific solutions like physics equations or molecule structures; concurrently, simulations operate as platforms for experimentation, offering observational feedback and optimizing continuous components like physical parameters. We focused on two scientific problems: constitutive law search and molecular design. Our approach outperforms other LLM-based benchmark methods, delivering consistent, robust, and nearly monotonic improvement. Furthermore, it shows exceptional ability in identifying unknown, true constitutive laws and molecular structures. Remarkably, our system generates innovative solutions that, despite being unconventional, are deemed reasonable after being thoroughly analyzed by experts in their respective domains. We view our process as a trailblazer, establishing a new paradigm to further advancements in physical scientific discoveries.

Chapter 5

Conclusion

This thesis aims to demonstrate the significant potential of physics-based world models and their construction using neural physics. Through the introduction of Rendering-Invariant State Predictor (RISP), Neural Constitutive Laws (NCLaw), and Scientific Generative Agent (SGA), this thesis simultaneously illustrates three key points: (1) the valuable functionality of physics-based world models, (2) the efficiency of constructing these models using differentiable neural physical modules, and (3) the substantial potential for their application to downstream scientific tasks.

RISP significantly enhances the utility of differentiable physics by introducing a neural module that operates on rendering outputs, thereby dramatically reducing the sim-to-real gap of the underlying world model. It addresses the longstanding question regarding the optimal utility of differentiable physics and proposes an elegant solution for training the neural module. RISP uniquely exhibits the surprising potential of neural physics, extending beyond mere physical modeling to achieving next-level efficiency and realism in world models.

Following RISP's demonstration of neural physics' efficiency and effectiveness in perception modeling of world models, NCLaw progresses further by parameterizing the constitutive laws, the core mechanism differentiating materials, using meticulously designed neural networks. NCLaw unifies representations for a variety of complex physical phenomena, ranging from elastic bodies to weakly compressible fluids. To ensure physical properties and stabilize training, several essential invariances, including rotational equiv-

ariance and static-state equilibrium, are embedded into the neural architecture. NCLaw paves the way for more accurate sim-to-real calibration and more efficient physics-based, data-driven simulations.

SGA builds directly on NCLaw's infrastructure but substitutes manually designed neural modules with tokens produced by large language models. Instead of directly optimizing world models to align with a physical trajectory based on neural parameters, differentiable simulation and language models are employed as a bilevel optimizer to update both the discrete expression and continuous parameterization of the physical laws. In addition to addressing the constitutive law searching issue in NCLaw, SGA explores its cross-disciplinary capacity in molecular design challenges. The experiments highlight SGA's superior performance in scientific discovery, suggesting a potential path toward fully automated scientific and engineering advancements.

5.1 Key Insights

Through building world models with neural physics using the discussed three methods, we conclude the lessons learned from the thesis.

- Embedding neural modules reasonably within the pipeline of physical world models enhances their efficiency, accuracy, and generalizability. Such strategic integration leverages the strengths of neural networks while complementing the traditional modeling techniques, especially for the components that are traditionally more empirically designed.

- In neural physics, designing appropriate invariance or equivariance is crucial for enhancing performance and stabilizing training. Additionally, given the ubiquitous presence of partial differential equations in neural physics, utilizing efficient higher-order operators is beneficial for modeling these invariances.

- Physical laws that have been extensively verified to be reliable and less error-prone should be preserved in their original form rather than being replaced through parameterization. For instance, maintaining the inherent connectivity between different

modules ensures that these world models do not have to incorporate unnecessary priors, thereby enhancing their robustness and efficiency.

- The development of physical world models offers advantages both within the realm of physical modeling and beyond it. The interaction between agents and these world models facilitates more efficient scientific discovery and fosters the evolution of intelligence, leading to deeper insights and advancements in various fields.

5.2 Limitations and Future Directions

While this thesis demonstrates both the methodology and potential of building world models with neural physics, it is essential to highlight the limitations of the current neural physics-based world models. These limitations naturally suggest avenues for future research and development.

- **Limited Modularization.** Modularization is an essential aspect of modern systems. While the work presented in the thesis follows high-level modularization principles, it constructs world models by fully replacing specific modules with a neural module, which is not explicitly modularized. Moreover, these efforts primarily target replacing the perceptual module and constitutive laws, neglecting other vital components of a physical system. These components include, but are not limited to, contact models, boundary conditions, spatial and time integrators, and geometric representations. Future research should consider exploring these additional modularizations.

- **Incomplete Interface to Multi-Modal Representation.** One of the key features of a world model is its capability for multi-modal extensions. A comprehensive and effective world model projects its latent dynamics into a full spectrum of representations, including visual and audio formats. RISP utilized differentiable physical rendering to access visual representations; however, this approach is both inefficient and restricted in scope. There is substantial potential in exploring the integration of dynamical models with NeRF [122], Gaussian splatting [83], video generative models [17], or other

multi-modal modules. This integration could enhance the efficiency and breadth of multi-modal representation capabilities within the world model.

- **Inefficient Data Acquisition.** The size and quality of data are key factors influencing scalability. Neural physics helps reduce the volume of data necessary for training world models by enforcing physically correct connections and invariances. However, this approach also increases the quality requirement to the state level. RISP extends this supervision from the state level to the pixel level by incorporating differentiable renderers. Despite this advancement, optimizing RISP still depends on state-level annotations. Future research could potentially explore methods such as self-supervised data acquisition using contrastive learning [25] or leveraging foundational vision models like DINOv2 [132, 29] or SAM [85]. These approaches might help improve scalability while maintaining or enhancing data quality.

- **Underexplored Steerability and Controllability.** Steering and controlling the evolution of world models pose significant challenges during both pre-training and post-training phases. In line with traditional physical simulations, world models built with neural physics generally incorporate interactions through boundary conditions and collisions. While these methods offer precise and comprehensive motion descriptions, they tend to be less user-friendly and creative. Recent advancements in conditional image generation have introduced new possibilities in this area through techniques such as text prompting [63] and visual prompting [208]. Integrating these modules into world models could be an exciting development, offering increased versatility and user interaction in the modeling process.

Appendix A

Appendix for RISP: Rendering-Invariant State Predictor with Differentiable Simulation and Rendering for Cross-Domain Parameter Estimation

A.1 Proof of the Theorem

For brevity, we remove the summation in $\mathcal{L}^{\text{reg}}$ and drop the index j to focus on deriving the gradients of the Frobenius term with respect to the network parameters $\boldsymbol{\theta}$. Let $\mathbf{G}$ be the Jacobian matrix inside the Frobenius norm, and let i and j be its row and column indices.

We now derive the gradient of $\mathcal{L}^{\text{reg}}$ with respect to the k-th parameter in $\boldsymbol{\theta}$ as follows:

$$\frac{\partial \mathcal{L}^{\text{reg}}}{\partial \boldsymbol{\theta}_k} = \sum_{ij} 2\mathbf{G}_{ij} \frac{\partial \mathbf{G}_{ij}}{\partial \boldsymbol{\theta}_k} \tag{A.1}$$

$$= \sum_{ij} 2\mathbf{G}_{ij} \frac{\partial}{\partial \boldsymbol{\theta}_k} [\frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I})_i}{\partial \mathbf{I}} : \frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}], \tag{A.2}$$

$$= \sum_{ij} 2\mathbf{G}_{ij} [\frac{\partial^2 \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I})_i}{\partial \mathbf{I} \partial \boldsymbol{\theta}_k} : \frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}] \tag{A.3}$$

$$= \sum_{i} \frac{\partial^2 \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I})_i}{\partial \mathbf{I} \partial \boldsymbol{\theta}_k} : (\sum_{j} 2\mathbf{G}_{ij} \frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}) \tag{A.4}$$

$$= \sum_{i} \frac{\partial^2 \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I})_i}{\partial \mathbf{I} \partial \boldsymbol{\theta}_k} : 2 \sum_{j} (\frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I})_i}{\partial \mathbf{I}} : \frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}) \frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}. \tag{A.5}$$

The derivation suggests that we can loop over the state dimension (indexed by i) and run backpropagation to obtain $\frac{\partial \mathcal{L}^{\text{reg}}}{\partial \boldsymbol{\theta}}$, assuming that all $\frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}$ are available. In fact, $\frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}$ depends on the training data only and remains constant throughout the whole optimization process, so we can pre-compute them in the training set, occupying an extra space of $\mathcal{O}(\sum |\boldsymbol{\psi}||\mathbf{I}|)$. Typically, $|\boldsymbol{\psi}_j|$ is a small number (less than 10 in most of our experiments), so it is affordable to store it with the images generated in the data set. This is also why we prefer a differentiable renderer that offers forward mode differentiation.

Assuming all $\frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}$ are ready to use, we can implement Eqn. (A.5) by two backpropagations for each state dimension i. First, backpropagating through $\mathcal{N}$ to obtain $\frac{\partial \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I})_i}{\partial \mathbf{I}}$. Next, we use $\frac{\partial \mathbf{I}}{\partial \boldsymbol{\psi}_j}$ to assemble the adjoint vector (the second summation in Eqn. (A.5)). Finally, we use the adjoint vector to backpropagate again through the second-order partial derivatives $\frac{\partial^2 \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{I})_i}{\partial \mathbf{I} \partial \boldsymbol{\theta}}$. Note that there is no need for looping over k in the second backpropagation. The total time cost is therefore $\mathcal{O}(|\mathbf{s}||\boldsymbol{\theta}|)$.

A.2 Implementation Details

A.2.1 Environments

Quadcoptor

State space We define the physical system of **quadcoptor** as a rigid body system without contact. We explicitly mark a task as "failed" when the center of the quadcoptor has an Euclidean distance to the origin $(0, 0, 0)$ of over 1000 at anytime. We define the state of the quadcoptor as its world position (x, y, z) and rotation angles $(\mathrm{yaw}, \mathrm{pitch}, \mathrm{roll})$. To alleviate the negative impact from the discontinuity of rotation angles, we design the output of the network as:

$$(x, y, z, \sin(\mathrm{yaw}), \sin(\mathrm{pitch}), \sin(\mathrm{roll}), \cos(\mathrm{yaw}), \cos(\mathrm{pitch}), \cos(\mathrm{roll})), \tag{A.6}$$

and later restore the rotation angles by $\mathrm{atan2}$.

Action space The control signals are 4-d representing the torques applied on 4 propellers. The magnitude of control signals is clamped to $[-500, 500]$.

Parameter space In system identification experiment, we uniformly sample the mass of the quadcoptor within $[2.8, 3.2)$. The ground-truth value of the mass is 3.

Cube

State space We define the physical system of **cube** as a rigid body system with contact. Similar to **quadcoptor** environment, we design the output of the network as a 9-d vector following A.6.

Parameter space In **cube** environment, we adopt a soft contact model paramterized by the spring stiffness k_n. In system identification experiment, we train $\lambda_k = \log_{10} k_n$ for better numerical stability and performance. We uniformly sample λ_k within $[5.5, 6.5)$. The ground-truth value of λ_k is 6.

Hand

State space We define the physical system of **hand** as an articulated body system powered by [199]. We fix the palm on the world frame and model the articulations as revolve joints. There are 13 joints in this environment, and each joints is clamped within $[-\frac{\pi}{4}, \frac{\pi}{4}]$. We define the state space of **hand** as a 13-d vector representing the angles of each joint $\{\theta_{1...13}\}$.

Action space The hand is controlled by applying torque on each joint. We paramterize the control signal of each finger at time t by:

$$u_t = m_j \sin\left(2\pi \min\left(1.0, \max\left(0.0, \frac{1}{T}\left(t - \delta_j\right)\right)\right)\right),$$

where $0 \leq m_j \leq 1$ and $0 \leq \delta_j \leq 40$ denote the action magnitude and action bias of finger j, and $T = 40$ is the period.

Parameter space For each finger, we let the joint stiffness of all joints on the finger available for training, making the parameter space a 5-d space. We uniformly sample the joint stiffness within $[0.0, 1.0)$. The ground-truth values of the joint stiffness are all 1.0.

Rod

State space We define the physical system of **rod** as a soft-body dynamics system powered by [37]. We fix both ends of the rod and apply a gravity force on it. We define a 10-d state space by evenly selecting 10 sensors along the central spine and track their positions along the vertical direction.

Action space We apply an external force on the center of the rob vertically as the action signal. We clamp the action within $[-500, 500)$.

Parameter space. We parameter the **rod** environment by Young's modulus E. We train $\lambda_E = \log_{10} E$ for better numerical stability and performance. We uniformly sample λ_E within $[4, 7)$ where the ground-truth value of λ_E is 5.

A.2.2 The RISP Network

Network setting We build the network upon a modified version of ResNet-18 [60]. All layers are followed by instance normalization [177] without affine parameters, following James et al. [72]. Since ReLU degenerates second derivatives to constant values, we replace it with *Swish* non-linearities [147]. We change the last layer of ResNet-18 backbone to a linear projection layer with the same number of output with the state dimension.

Training We use *Adam* optimizer [84] with a learning rate of 1e-2 and a weight decay of 1e-6 for training. The learning rate has a cosine decay scheduler [111]. We train the network for 100 epochs. We set the batch size to 16 by default. To avoid model collapse in the early stage of training, we linearly increase the magnitude coefficient of the rendering gradients from 0 to a environment-specific value: we set it to 1 for **quadrotor**, 20 for **cube**, 100 for **hand**, and 30 for **rod**.

A.2.3 Training Details

We share the same experiment settings across imitation learning, and visuomotor control experiments experiments. We use RMSProp optimizer with a learning rate of 3e-3 and a momentum of 0.5. We optimize the system parameters or the action parameters by 100 iterations.

A.3 More Experiments

A.3.1 Visuomotor Control

We consider a visuomotor control task defined as follows: given a target image displaying the desired state of the physical system, we optimize a sequence of actions that steer the physical system to the target state from a randomly generated initial state. We set the target states by selecting them from the ground truth in the imitation learning tasks. We report in Table A.1 the state error computed from experiments repeated with various rendering configurations and initial guesses. The state error is defined as L1 distances between the

desired state and the final state from the simulator. The smaller state error and standard deviation from our methods in Table A.1 shows the advantages of our approach over other baselines. As before, the performance from all methods is capped by the oracle, which requires much more knowledge about the ground-truth state.

A.3.2 Real-World Experiment

Problem Setup

We consider an imitation learning task for the real-world quadrotor. The imitation learning takes as input the intrinsic and extrinsic parameters of the camera, a real-world quadrotor video clip recorded by the camera, empirical values of the quadrotor's system parameters, and the geometry of the quadrotor represented as a mesh file. We aim to recover an action sequence so that the simulated quadrotor resembles the motion in the input video as closely as possible.

Experimental Setup

Video recording To obtain a real-world video clip, we set up a calibrated camera with known intrinsic and extrinsic parameters. During the whole recording procedure, the camera remains a fixed position and pose. After we obtain the original video clip, we trim out the leading and tailing frames with little motion and generate the pre-processed video clip with total length of 14.4s and frequency of 10Hz.

	quadrotor	**hand**	**rod**
average	1.38 ± 0.00	0.23 ± 0.00	0.89 ± 0.00
random	63.93 ± 9.44	0.32 ± 0.06	0.87 ± 0.06
pixelwise-loss	3.02 ± 1.35	0.25 ± 0.05	0.05 ± 0.01
perceptual-loss	1.59 ± 0.26	0.28 ± 0.05	0.11 ± 0.06
ours-no-grad	$1.30 \pm \mathbf{0.15}$	0.23 ± 0.06	$\mathbf{0.04} \pm \mathbf{0.0002}$
ours	$\mathbf{0.54} \pm 0.25$	$\mathbf{0.11} \pm \mathbf{0.02}$	$\mathbf{0.04} \pm 0.0006$
oracle	0.15 ± 0.09	0.002 ± 0.00	0.01 ± 0.02

Table A.1: Visuomotor control results. Each entry reports the mean and standard deviation of the state discrepancy computed from 4 randomly generated initial guesses and rendering conditions.

Quadrotor To replicate a potential deployment, we choose a commercial quadrotor that is publicly available to users. We manually control the quadrotor so that it follows a smooth trajectory.

Motion capture (MoCap) system To evaluate the performance of various methods, we also build a MoCap system to log the position and rotation of the quadrotor. We attach six reflective markers to the quadrotor so that the MoCap system tracks the local coordinate system spanned by these markers. Note that we only use MoCap data for test use, it is not necessarily a step in our pipeline, and none of our methods ever used the MoCap data as a dependency.

Network and dataset details To be consistent with other experiments, we use the same network architecture for RISP as the **quadrotor** environment. We uniformly sample the position and pose ensuring that the quadrotor is observable on the screen. Every sampled state comes with a different rendering configuration. We generate in total 10k data points and divide them by 8:2 for training and testing.

Imitation Learning

Here we perform the imitation learning task where we reconstruct the action sequence from the real-world video clip. This task is much more challenging than the simulated ones. First, the real-world physics has unpredictable noises that is impossible to be modeled perfectly by the differentiable simulation. Additionally, the real-world video has unseen appearances including but not limited to the lighting source and the floor texture. It is also noteworthy that the real-world video is exceedingly long which increases the dimension of action sequence dramatically.

In order to solve these challenges, we design a slightly different pipeline of imitation learning for the real-world experiments using RISP. We first run RISP on the input video to generate a target trajectory of the quadrotor. Next, we minimize a loss function defined as the difference between the target trajectory and a simulated trajectory from a differentiable quadrotor simulator, with the action at each video frame as the decision variables. Because

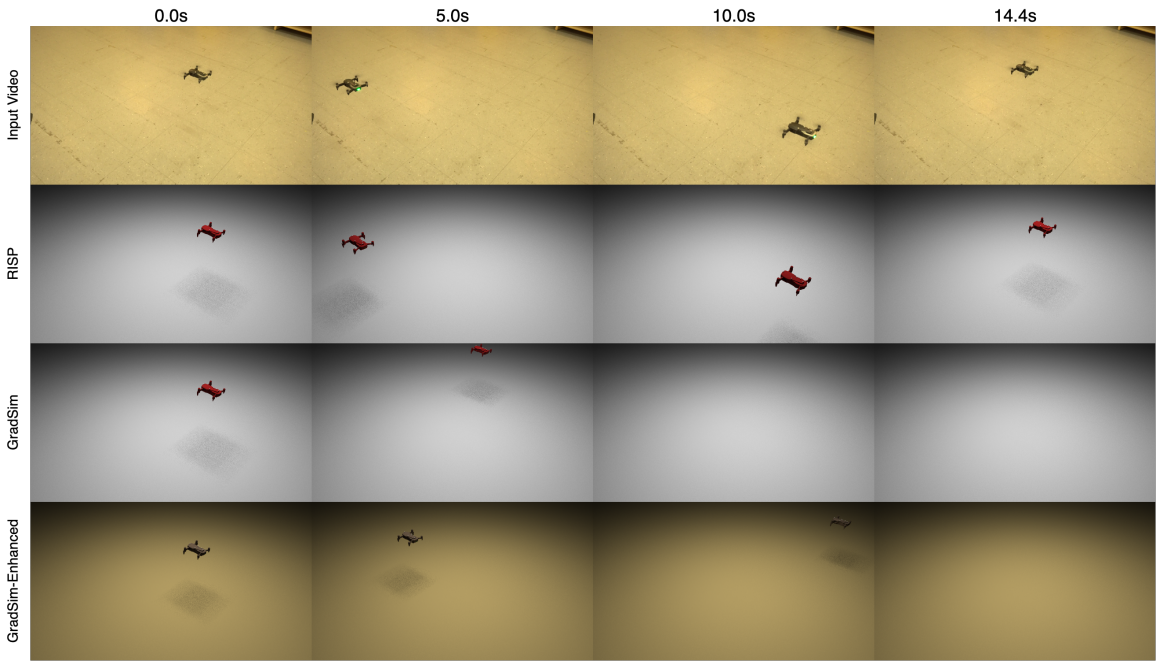


Figure A-1: Imitation learning in the real-world experiment. Given a reference video (top), the goal is to reconstruct a sequence of actions that resembles its motion. We illustrate the motions reconstructed using our method (upper middle), the pixelwise loss (lower middle), and its enhanced variant (bottom).

this optimization involves many degrees of freedom and a long time horizon, we find a good initial guess crucial. We initialize the action sequence using the output of a handcrafted controller that attempts to follow the target trajectory. Such an initial guess ensures the quadrotor stays in the camera's view and is used by our method and the baselines.

Results

We show the results of the real-world experiment in Fig. A-1 by visualizing a representative subset of frames.

RISP To validate the robustness of our method, we aggressively randomized the rendering configuration so that it differs significantly from the input video. Our method reconstructs a similar dynamic motion without access to the rendering configuration or the exact dynamic model in the input video.

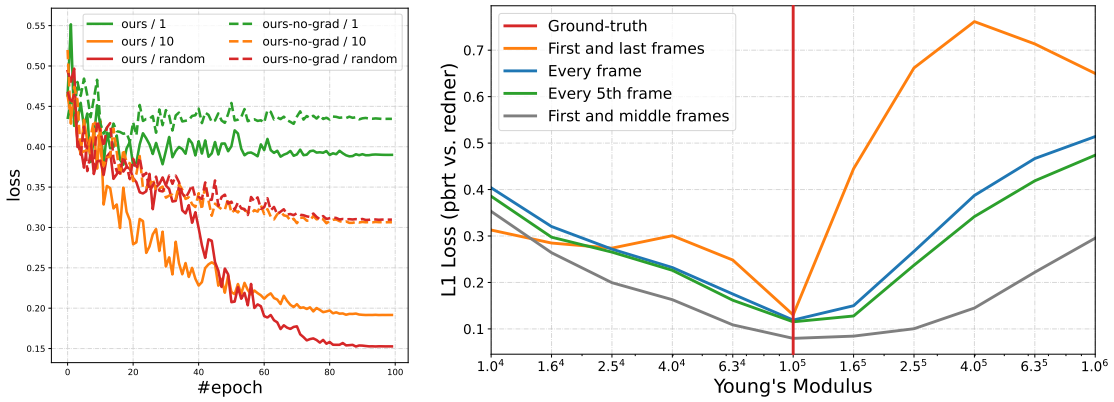
GradSim We compare the pixelwise loss from ∇Sim [126] with our method by sharing the identical rendering configuration and initial guesses. We observe that the pixelwise loss does not provide valid guidance to the optimization and steers the quadrotor out of the screen immediately. We presume the major reason behind the degeneration of pixelwise loss is the assumption of a known rendering configuration in the target domain. However, such information is generally non-trivial to obtain from a real-world video.

GradSim-Enhanced According to our presumption above, we propose two modifications for improvement: First, we manually tuned the rendering configuration until it appears as close as possible to the real-world video, and second, we provide it with a good initial guess of the action sequence computed based on the ground-truth trajectory recorded from a motion capture system. Note that the good initial guess from motion capture system is generally impossible to access in real-world applications and is not used in **RISP** or **GradSim**. Even with these strong favors, **GradSim-Enhanced** only recovers the very beginning of the action sequence and ends up with an uncontrollable drifting out of screen.

A.3.3 Ablation Study

Rendering Gradients

We study the impact of rendering gradients on the data efficiency by the comparison between **ours** and **ours-no-grad**. We reuse the state estimation data sets (Table 2.1) on **quadrotor** but vary the number of rendering configurations between 1 and 10. We then train both of our methods for 100 epochs and report their performances on a test set consisting of 200 randomly states, each of which is augmented by 10 unseen rendering configurations. We show the result on Fig. A-2a. The right inset summarizes the performances of **ours** (solid lines) and **ours-no-grad** (dashed lines) under varying number of rendering configurations, with the green, orange, and red colors corresponding to results trained on 1, 10, and randomly sampled rendering configurations. It is obvious to see that all solid lines reach a lower state estimation loss than their dashed counterparts, indicating that our rendering gradient digs more information out of the same amount of rendering configurations. It is



(a) **(b)**

Figure A-2: (a) The number of epoch versus loss curves. The solid and dashed curves represent the training curves of ours and ours-no-grad respectively. The color of the lines indicates the number of rendering configurations in training set where green and orange are 1 and 10, and red curve samples a different rendering configuration for every training data. (b) The loss versus Young's modulus curves. The orange, blue, green, and gray curves represent the loss landscapes with respect to the Young's modulus given different temporal sampling rates. The red vertical line shows where the ground-truth Young's modulus locates.

worth noting that with only 10 rendering configurations (orange solid line), our method with rendering gradients achieves a lower loss than the one without but using randomly sampled rendering configurations (red dashed line), which reflects the better data efficiency.

By comparing **ours** and **ours-no-grad** from Table 2.1, 2.2, 2.3, and A.1, we can see that having the rendering gradients in our approach is crucial to its substantially better performance. We stress that having a rendering-invariant state estimation is the core source of generalizability in our approach and the key to success in many downstream tasks.

Temporal Sampling Rate

We study the impact of temporal sampling rate on the performance of RISP using four different sampling strategies: sampling densely on every frame, sampling sparsely on every 5th frame, sampling only on the first and last frames, and sampling only on the first and middle frames. We reuse the system identification task in **rod** environment and plot the loss landscapes of them around the ground-truth Young's Modulus in Fig. A-2b. We observe that RISP is robust against different sampling rates due to the same global optima of all

curves. Additional, except for the extreme case with substantial information loss (first and last frames), most of the curves are unimodal with one local minimum indicating an easier problem in optimization. Thus, we believe it is safe to expect our method with a standard gradient-based optimizer to succeed under various temporal sampling rates.

Appendix B

Appendix for Learning Neural Constitutive Laws from Motion Observations for Generalizable PDE Dynamics

B.1 The Material Point Method

This section's goal is to demonstrate how MPM (i.e., the time integration scheme in $\mathcal{I}$ from Algorithm 1) is systematically derived from the governing PDEs. Additional derivation details can be found in the works by Sulsky et al. [164], Stomakhin et al. [162], Yue et al. [206], Jiang et al. [76, 75].

To derive MPM, we start with the Eulerian forms for the conservation of mass and conservation of momentum,

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \boldsymbol{v}, \tag{B.1}$$

$$\rho \boldsymbol{a} = \nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{b}, \tag{B.2}$$

where ρ is the density, $\boldsymbol{a}$ is the acceleration, $\boldsymbol{\sigma}$ is the Cauchy stress tensor, and $\boldsymbol{b}$ is the body

force. Note that mass conservation (Eqn. (B.1)) does not require special treatments since it is automatically satisfied by advecting Lagrangian MPM particles.

B.1.1 Weak Form

We obtain the weak form of the equation of motion (Eqn. (B.2)) by multiplying a test function $\boldsymbol{w}$ on both sides and integrating it over an arbitrary domain Ω:

$$\int_\Omega \rho \boldsymbol{a} \cdot \boldsymbol{w} d\Omega = \int_\Omega (\nabla \cdot \boldsymbol{\sigma}) \cdot \boldsymbol{w} d\Omega + \int_\Omega \rho \boldsymbol{b} \cdot \boldsymbol{w} d\Omega, \tag{B.3}$$

This should be satisfied for any test function $\boldsymbol{w}$ and any domain Ω.

With integration by parts, the equation above becomes

$$\begin{aligned} \int_\Omega \rho \boldsymbol{a} \cdot \boldsymbol{w} d\Omega = -\int_\Omega \boldsymbol{\sigma} : \nabla \boldsymbol{w} d\Omega + \int_{\partial\Omega_{\boldsymbol{T}}} \boldsymbol{w} \cdot \boldsymbol{T} dS \\ + \int_\Omega \rho \boldsymbol{b} \cdot \boldsymbol{w} d\Omega, \end{aligned} \tag{B.4}$$

where $\boldsymbol{T}$ is the traction and $\partial\Omega_{\boldsymbol{T}}$ is the boundary where traction is applied.

B.1.2 Spatial Discretization

With two sets of basis functions, one for the material points and the other for the grid, MPM discretizes the weak form spatially and obtains the following discretized system:

$$\begin{aligned} \sum_{b=1}^{G} M_{ab} \boldsymbol{a}_b = -\sum_{i=1}^{Q} V_i^0 \boldsymbol{\tau}_i \nabla N_a(\boldsymbol{x}_i) \\ + \sum_{i=1}^{Q} M_i N_a(\boldsymbol{x}_i) \boldsymbol{b}_i, \end{aligned} \tag{B.5}$$

where $M_{ab} = \sum_{i=1}^{Q} M_i N_a(\boldsymbol{x}_i) \cdot N_b(\boldsymbol{x}_i)$ is the full mass matrix; $V_i^0, M_i, \boldsymbol{\tau}_i, \boldsymbol{x}_i, \boldsymbol{b}_i$ are the initial volume, mass, Kirchhoff stress, current position, the external force of material point i; Q is the number of material points; $N_b, \boldsymbol{a}_b$ are the basis function and acceleration on the Eulerian grid node b; G is the number of Eulerian basis functions. We refer to Sulsky et al.

[164] for additional details on this derivation.

B.1.3 Temporal Discretization

Using the explicit Euler scheme with a time step size Δt, we can discretize in time,

$$\sum_{b=1}^{G} M_{ab} \frac{\boldsymbol{v}_b^{n+1} - \boldsymbol{v}_b^n}{\Delta t} = -\sum_{i=1}^{Q} V_i^0 \boldsymbol{\tau}_i^n \nabla N_a(\boldsymbol{x}_i^n) + \sum_{i=1}^{Q} M_i N_a(\boldsymbol{x}_i^n) \boldsymbol{b}_i^n. \tag{B.6}$$

B.1.4 MPM Pseudocode

From these spatial and temporal discretizations, we arrive at the pseudocode in Algorithm 4. We implement this algorithm under the MLS-MPM framework by Hu et al. [65].

This completes the background on MPM and the time integration scheme in $\mathcal{I}$ from Algorithm 1. Note that neural constitutive laws introduce in our work contribute to time integration in two ways. First, the neural elastic constitutive law $\mathcal{E}_{\theta_e}$ computes the first Piolar-Kirchhoff stress $\boldsymbol{P}$ (equivalently, the Cauchy stress $\boldsymbol{\sigma}$ or the Kirchhoff stress $\boldsymbol{\tau}$). This stress is used for computing the forces of the grid node. Second, the neural plastic constitutive law $\mathcal{P}_{\theta_p}$ post-processes the trial elastic deformation gradient by projecting back to the admissible elastic region.

B.2 Material Models for Training

Below we list the material models for generating ground truth *training data*. For each material, we list both the elastic constitutive law ($\mathcal{E}_\mu$) and the plastic constitutive law ($\mathcal{P}_\mu$).

We emphasize that our method can capture all these expert-designed classic constitutive models using the same network architecture and the same training strategy.

Algorithm 4 MPM Algorithm

Input: Position $\boldsymbol{x}_n^i$, velocity $\boldsymbol{b}_n^i$, and elastic deformation gradient $\boldsymbol{F}_n^{e,i}$ of each material point, $i = 1, \ldots, Q$ at time instance t_n

Output: Position $\boldsymbol{x}_{n+1}^i$, velocity $\boldsymbol{b}_{n+1}^i$, trial elastic deformation gradient $\boldsymbol{F}_{n+1}^{e,\text{trial},i}$, $i = 1, \ldots, Q$ at time instance t_{n+1}

1: Transfer Lagrangian kinematics to the Eulerian grid by performing a "particle to grid" transfer: Compute for $b = 1, \ldots, G$

$$
\begin{aligned}
m_{b,n} &= \sum_{i=1}^{Q} N_b(\boldsymbol{x}_n^i) M_i \\
m_{b,n}\boldsymbol{b}_{b,n} &= \sum_{i=1}^{Q} N_b(\boldsymbol{x}_n^i) M_i \boldsymbol{b}_n^i \\
\boldsymbol{f}_{b,n}^{\boldsymbol{\sigma}} &= -\sum_{i=1}^{Q} \frac{J(\boldsymbol{F}_n^{e,i})}{\rho_0} \boldsymbol{\sigma}(\boldsymbol{F}_n^{e,i}) \nabla N_b(\boldsymbol{x}_n^i)\ M_i \\
\boldsymbol{f}_{b,n}^{e} &= \sum_{i=1}^{Q} \frac{J(\boldsymbol{F}_n^{e,i})}{\rho_0} \boldsymbol{b}(\boldsymbol{x}_n^i) N_b(\boldsymbol{x}_n^i)\ M_i
\end{aligned}
$$

2: Solve Eulerian governing equations by computing for $b = 1, \ldots, G$

$$
\begin{aligned}
\dot{\boldsymbol{b}}_{b,n+1} &= \frac{1}{m_{b,n}} (\boldsymbol{f}_{b,n}^{\boldsymbol{\sigma}} + \boldsymbol{f}_{b,n}^{e}) \\
\Delta \boldsymbol{b}_{b,n+1} &= \dot{\boldsymbol{b}}_{b,n+1} \Delta t \\
\boldsymbol{b}_{b,n+1} &= \boldsymbol{b}_{b,n} + \Delta \boldsymbol{b}_{b,n+1}
\end{aligned}
$$

3: Update the Lagrangian velocity and deformation gradient by performing a "grid to particle" transfer: Compute for $i = 1, \ldots, Q$

$$
\begin{aligned}
\boldsymbol{b}_{n+1}^i &= \sum_{i=1}^{G} N_i(\boldsymbol{x}_n^i) \boldsymbol{b}_{b,n+1} \\
\boldsymbol{F}_{n+1}^{e,\text{trial},i} &= (\boldsymbol{I} + \sum_{i=1}^{G} \boldsymbol{b}_{i,n+1} \otimes \nabla N_i(\boldsymbol{x}_n^p) \Delta t) \boldsymbol{F}_n^{e,i}
\end{aligned}
$$

4: Update Lagrangian positions for $i = 1, \ldots, Q$

$$
\boldsymbol{x}_{n+1}^i = \boldsymbol{x}_n^i + \Delta t \boldsymbol{b}_{n+1}^i
$$

Purely Elastic Examples of purely elastic materials include rubber, biological tissues, and jelly. We use the fixed corotated hyperelastic model by Stomakhin et al. [161].

$$
\boldsymbol{P} = 2\mu(\boldsymbol{F} - \boldsymbol{R}) + \lambda J(J-1)\boldsymbol{F}^{-T}, \tag{B.7}
$$

where $\boldsymbol{R}$ is the rotation matrix from the polar decomposition of $\boldsymbol{F} = \boldsymbol{RS}$. There is no plasticity in this material. Equivalently, the plastic return mapping is an identity map,

$$\mathcal{P}(\boldsymbol{F}) = \boldsymbol{F}. \tag{B.8}$$

Elastic with the Drucker-Prager Yield Condition We use the Saint Venant–Kirchhoff elastic model. Computing the singular value decomposition of $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ and the Hency strain $\boldsymbol{\epsilon} = \log(\boldsymbol{\Sigma})$, we have [11]

$$\boldsymbol{P} = \boldsymbol{U}(2\mu\boldsymbol{\epsilon} + \lambda \operatorname{Tr}(\boldsymbol{\epsilon}))\boldsymbol{U}^T. \tag{B.9}$$

In addition, we apply the Drucker-Prager yield condition [34, 86, 207, 23]:

$$\operatorname{Tr}(\boldsymbol{\epsilon}) > 0 \quad \text{or} \quad \delta\gamma = \|\hat{\boldsymbol{\epsilon}}\| + \alpha\frac{(3\lambda + 2\mu)\operatorname{Tr}(\boldsymbol{\epsilon})}{2\mu} > 0, \tag{B.10}$$

where $\alpha = \sqrt{\frac{2}{3}}\frac{2\sin\theta}{3-\sin\theta}$ and θ is the friction angle of the granular media.

With this yield criterion, we then define two types of plastic projection. When the material undergoes tension, we project the stress unto the cone tip; When the material undergoes compression and violates the cone constraint, we project the stress onto the cone in an isochoric manner. Mathematically, we have

$$\mathcal{P}(\boldsymbol{F}) = \begin{cases} \boldsymbol{U}\boldsymbol{V}^T & \operatorname{Tr}(\boldsymbol{\epsilon}) > 0, \\ \boldsymbol{F} & \operatorname{Tr}(\boldsymbol{\epsilon}) \le 0 \ \ \& \ \ \delta\gamma \le 0, \\ \boldsymbol{U}\exp(\epsilon - \delta\gamma\frac{\hat{\boldsymbol{\epsilon}}}{\|\hat{\boldsymbol{\epsilon}}\|})\boldsymbol{V}^T & \operatorname{Tr}(\boldsymbol{\epsilon}) \le 0 \ \ \& \ \ \delta\gamma > 0. \end{cases} \tag{B.11}$$

Elastic with the von Mises Yield Condition We adopt the same Saint Venant–Kirchhoff elastic model detailed previously,

$$\boldsymbol{P} = \boldsymbol{U}(2\mu\boldsymbol{\epsilon} + \lambda \operatorname{Tr}(\boldsymbol{\epsilon}))\boldsymbol{U}^T. \tag{B.12}$$

However, in this case, we adopt the von Mises yield condition [123, 65, 70],

$$\delta\gamma = \|\hat{\boldsymbol{\epsilon}}\| - \frac{\tau_Y}{2\mu}, \tag{B.13}$$

where $\boldsymbol{\epsilon}$ is the normalized Hencky strain computed from the deformation gradient $\boldsymbol{F}$. τ_Y is the yield stress and describes how easily the material undergoes the plastic flow. A positive $\delta\gamma$, i.e., $\delta\gamma > 0$, suggests the material violates the yield constraint. For stress that violates the yield criterion, we will project it back into the elastic region via an isochoric (volume-preserving) projection:

$$\mathcal{P}(\boldsymbol{F}) = \begin{cases} \boldsymbol{F} & \delta\gamma \leq 0, \\ \boldsymbol{U}\exp(\epsilon - \delta\gamma\frac{\hat{\boldsymbol{\epsilon}}}{\|\hat{\boldsymbol{\epsilon}}\|})\boldsymbol{V}^T & \delta\gamma > 0. \end{cases} \tag{B.14}$$

Weakly Compressible Fluids Following Stomakhin et al. [163], Tampubolon et al. [167], we model fluids using the aforementioned fixed corotated elastic model with $\mu = 0$, effectively modeling the fluid with no shearing resistance while penalizing any volume change,

$$\boldsymbol{P} = \lambda J(J-1)\boldsymbol{F}^{-T}. \tag{B.15}$$

In addition, we adopt the plasticity model by Stomakhin et al. [163], Gao et al. [46],

$$\mathcal{P}(\boldsymbol{F}) = J^{\frac{1}{3}}\boldsymbol{I}. \tag{B.16}$$

Effectively, this flow rule projects the deformation gradient onto the hydrostatic axis in an isochoric manner.

In this work, we focus on exploring the aforementioned four types of materials. We leave it as future work to explore other elastoplastic materials, e.g., non-Newtonian fluids [206].

Algorithm 5 Neural Constitutive Laws

Input: $\boldsymbol{F}$ ($\boldsymbol{F}^e$ for elasticity; $\boldsymbol{F}^{e,\text{trial}}$ for plasticity)
Output: $\boldsymbol{P}$ or $\boldsymbol{F}^{e,\text{new}}$

$\boldsymbol{F} \overset{\text{SVD}}{=} \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ // singular value decomposition
$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^T$ // rotation equivariant
$\boldsymbol{T}_1 = \text{NN}(\boldsymbol{\Sigma}, \boldsymbol{F}^T\boldsymbol{F}, \det(\boldsymbol{F}))$ // rotation invariant
$\boldsymbol{T}_2 = \frac{1}{2}(\boldsymbol{T}_1 + \boldsymbol{T}_1^T)$ // ensure symmetry
$\boldsymbol{Y} = \boldsymbol{R}\boldsymbol{T}_2$ // ensure rotation equivariance
if elasticity **then**
    $\boldsymbol{P} = \boldsymbol{Y}$
end if
if plasticity **then**
    $\boldsymbol{F}^{e,\text{new}} = \boldsymbol{F} + \alpha\boldsymbol{Y}$
end if

B.3 Algorithm of Neural Constitutive Laws

B.3.1 Algorithm

We demonstrate the algorithm pipeline in Algorithm 5. Note that we can easily implement polar decomposition that $\boldsymbol{F} = \boldsymbol{R}\boldsymbol{S}$ by singular value decomposition (SVD) that $\boldsymbol{F} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ and get $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^T$ and $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^T$. We use a highly-optimized 3×3 SVD implementation on GPUs [46]. We use $\alpha = 0.001$ in all our experiments.

B.3.2 Proof of Rotation Invariance

Proof. Our network takes three inputs, we prove they are rotation-invariant with respect to an arbitrary rotation $\boldsymbol{R}^* \in \text{SO}(3)$ one by one:

1. The singular values $\boldsymbol{\Sigma}$. We have:

$$\boldsymbol{F}_{\text{new}} = \boldsymbol{R}^*\boldsymbol{F} \tag{B.17}$$

$$= \boldsymbol{R}^*\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{B.18}$$

$$= (\boldsymbol{R}^*\boldsymbol{U})\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{B.19}$$

where $\boldsymbol{R}^*\boldsymbol{U}$ is still a unitary matrix because both $\boldsymbol{R}^*$ and $\boldsymbol{U}$ are unitary matrices. As a result, when $\boldsymbol{R}^*$ applied, $\boldsymbol{\Sigma}$ is invariant.

2. The right Cauchy-Green tensor $\boldsymbol{F}^T\boldsymbol{F}$. We have:

$$\boldsymbol{F}_{\text{new}}^T\boldsymbol{F}_{\text{new}} = (\boldsymbol{R}^*\boldsymbol{F})^T(\boldsymbol{R}^*\boldsymbol{F}) \tag{B.20}$$

$$= \boldsymbol{F}^T(\boldsymbol{R}^*)^T\boldsymbol{R}^*\boldsymbol{F} \tag{B.21}$$

$$= \boldsymbol{F}^T\boldsymbol{F}. \tag{B.22}$$

3. The determinant of the deformation gradient $\det(\boldsymbol{F})$. We have:

$$\det(\boldsymbol{F}_{new}) = \det(\boldsymbol{R}^*\boldsymbol{F}) \tag{B.23}$$

$$= \det(\boldsymbol{R}^*)\det(\boldsymbol{F}) \tag{B.24}$$

$$= \det(\boldsymbol{F}), \tag{B.25}$$

because the determinant of a unitary matrix is 1.

As a result, all inputs to the neural networks are rotation invariant, which implies the output of the neural networks is rotation invariant. We also have that $\mathrm{NN}(\boldsymbol{R}^*\boldsymbol{F}) = \mathrm{NN}(\boldsymbol{F})$, which makes both $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ rotation invariant since they solely depend on the output of neural networks. □

B.3.3 Proof of Rotation Equivariance

Proof. With an arbitrary rotation $\boldsymbol{R}^* \in \mathrm{SO}(3)$ applied, the input to neural constitutive laws will be transformed into $\boldsymbol{R}^*\boldsymbol{F}$. By the definition of polar decomposition:

$$\boldsymbol{F} \overset{\text{PD}}{=} \boldsymbol{R}\boldsymbol{S} \tag{B.26}$$

$$\boldsymbol{R}^*\boldsymbol{F} = \boldsymbol{F}_{\text{new}} \overset{\text{PD}}{=} \boldsymbol{R}_{\text{new}}\boldsymbol{S} = (\boldsymbol{R}^*\boldsymbol{R})\boldsymbol{S}. \tag{B.27}$$

As a result, given $\boldsymbol{F}_{\text{new}}$ as the input, the output of neural constitutive laws is

$$\boldsymbol{Y}_{\text{new}} = \boldsymbol{R}_{\text{new}}\boldsymbol{T}_2 = \boldsymbol{R}^*\boldsymbol{R}\boldsymbol{T}_2 = \boldsymbol{R}^*\boldsymbol{Y}, \tag{B.28}$$

which proves the rotation equivariance of $\boldsymbol{Y}$. For elasticity, the rotation equivariance

automatically inherits. For plasticity, we have:

$$\begin{aligned}
\boldsymbol{F}_{\text{new}}^{e,\text{new}} =& \boldsymbol{F}_{\text{new}}^{e,\text{trial}} + \alpha \boldsymbol{Y}_{\text{new}} && \text{(B.29)}\\
=& \boldsymbol{R}^* \boldsymbol{F}^{e,\text{trial}} + \alpha \boldsymbol{R}^* \boldsymbol{Y} && \text{(B.30)}\\
=& \boldsymbol{R}^* (\boldsymbol{F}^{e,\text{trial}} + \alpha \boldsymbol{Y}) && \text{(B.31)}\\
=& \boldsymbol{R}^* \boldsymbol{F}^{e,\text{new}}, && \text{(B.32)}
\end{aligned}$$

which proves the rotation equivariance of neural plasticity constitutive laws. □

B.4 Implementation Details

B.4.1 Simulation

For each training environment, we load material points inside a 0.5^3 m cube. We set the grids in the MPM simulator to be $20 \times 20 \times 20$ to speed up the simulation. We use a threshold of 3 grids to detect the collision and set a free-slip boundary condition within a 1.0^3 m box. Our simulation always applies a gravity of -9.8 m/s^2. We uniformly use the same hyper-parameters for MPM simulation for all our training environments. We implement our differentiable MPM simulator on GPUs using Warp [118].

B.4.2 Network and Training

Our neural constitutive laws contain two neural networks with equal sizes each for elasticity and plasticity, a total of 11,008 parameters. The neural networks use GELU [62] as non-linearity and contain no normalization layers. We use the Adam optimizer [84] with learning rates of 1.0 and 0.1 for elasticity and plasticity, respectively. We train the neural constitutive laws for 300 epochs and decay the learning rates of both elasticity and plasticity using a cosine annealing scheduler. We also clip the norm of gradients to a maximum of 0.1. We also utilize a "teacher-forcing" scheme that restarts from the ground-truth position periodically. We increase the period of teacher forcing from 25 to 200 steps by a cosine annealing scheduler. We train all our experiments on one NVIDIA RTX A6000. All

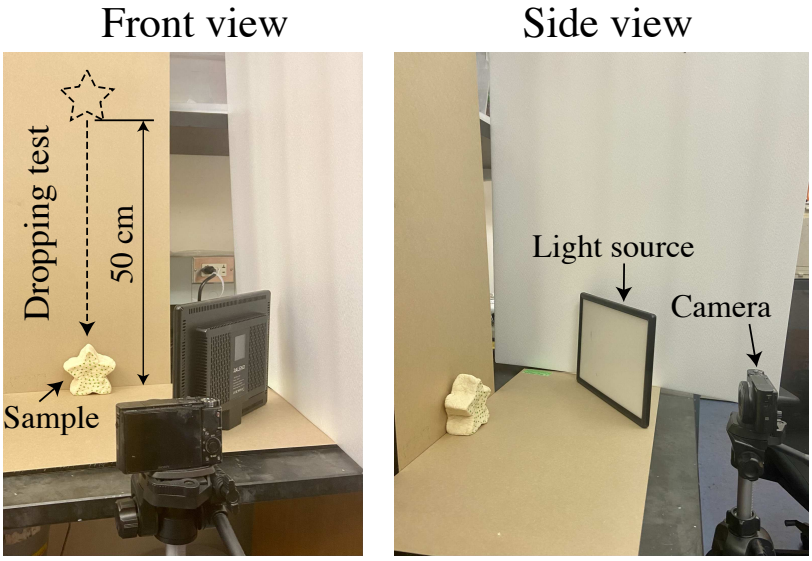


Figure B-1: Real Experiment. We illustrate the front and side view of our experiment setup. We indicate the dropping position of the dough with a dashed outline and the direction with a dashed arrow.

experiments share the same training configurations and hyper-parameters without ad-hoc tuning.

We implement the neural networks using PyTorch [134], which can share CUDA memory with Warp in order to reduce the overhead interacting with the simulator.

B.4.3 Real-World Experiment

We use dough as our testing material. Dough is a malleable and elastic paste made from a mixture of flour and water. Dough is famous for its complex and hard-to-predict mechanical properties [119, 3] and, therefore, a good test for our algorithm. Our dough is fabricated by mixing 125 g flour with 75 g water and kneading for around 5 minutes. A variety of 2D shapes are then formed out of the dough for training and testing purposes.

In physical experiments (See Fig. B-1), the dough is dropped from a height of 50 cm and collides with a rigid surface. This process is recorded by a high-speed camera (SONY RX100) at 960 frames per second with an exposure time of 1/4000 second. The dough is marked with around 50 green dots on the front surface, which are later tracked to provide precise information about the deformation history.

B.5 Extra Discussion

B.5.1 Generalization to Mesh-Based Simulation

Among all components in the physical simulation, our method only replaces the manually tuned constitutive laws with a carefully designed neural network, which is a data-driven representation with better expressiveness, uniformity, and versatility. Thus, neural constitutive laws are agnostic to discretization and simulation frameworks. In the current work, we select the material point method (MPM) as the backbone due to its compatibility with various materials, easing our implementation. That said, our framework is general and discretization-independent. We expect that our method also applies to mesh-based or hybrid mesh/particle simulation or any discretization framework. However, it will require significant engineering work to implement the supporting differentiable simulation for mesh and mesh/particle coupling, and we believe such an endeavor would be best served by its own dedicated manuscript.

B.6 Extra Experiments

B.6.1 Runtime Comparison

We evaluate the runtime performance of our method and compare it with baselines in Tab. B.1. We conduct this experiment by running for a certain period so that the simulation covers 0.5s in the real world. Because of the difference in time step size, it takes 1k steps for ours and analytical, while it takes 200 steps for gnn. We rerun the same trajectory 10 times and take the average to reduce the variance. Even though gnn takes much larger steps, we find our method run at a comparable speed with gnn. We suspect the reason is (1) we develop all components of our method on GPUs, and (2) the radius graph network is time-consuming. Since we did not focus on runtime optimization, there is still room for our method to improve, including (1) removing redundant SVD in elasticity and plasticity, (2) reducing the communication between Warp and PyTorch, and (3) replacing PyTorch with light-weight neural network libraries in deployment.

Table B.1: Runtime comparison.

Method	Jell-O	Sand	Plasticine	Water
ours	2.56s	2.55s	2.62s	2.56s
gnn	2.35s	2.10s	2.19s	2.26s

Table B.2: More recent comparison.

Task	**ours**	**gnn**	Li and Farimani [102]
reconstruction	**6.5e-5**	8.7e-3	9.3e-4
time	**1.4e-4**	1.1e-2	5.2e-3
velocity	**4.6e-5**	6.8e-3	1.0e-3
geometry	**2.3e-4**	3.7e-2	4.7e-2

B.6.2 More Recent Comparison

We select one of the related previous works [102] and evaluate it on a subset of our experiments. We use Plasticine as the environment and report the performance comparison on four tasks as shown in Tab. B.2. According to the results, Li and Farimani [102] is better than the original gnn baseline in our manuscript on 3 out of 4 tasks. However, our method is still stronger than these GNN-based methods by orders of magnitude. There are two potential reasons: (1) our method incorporates physics-aware network architectures ensuring rotation equivariance and undeformed state equilibrium, and (2) our method disentangles the learning of constitutive laws from the simulation framework and promotes data efficiency.

B.6.3 Parameter Identification

We generate a trajectory using Water and randomly select 1k deformation gradients as the input to the neural networks. We measure the mean squared errors (MSE) of ours and labeled versus the ground truth and report the results in Tab. B.3. Note that the error of $\boldsymbol{P}$ is naturally larger than deformation gradients because of the physical meaning of the stress tensor (often orders of magnitude larger than the deformation gradient). As shown in the table, our method achieves a similar result in $\boldsymbol{F}^{e,\text{new}}$ but falls behind in $\boldsymbol{P}$ compared to labeled. There are a few reasons behind it. First, labeled actually uses extra supervision

Table B.3: Parameter identification.

Term	ours	labeled
$\boldsymbol{P}$	6.1e-1	**2.6e-3**
$\boldsymbol{F}^{e,\text{new}}$	2.388e-6	**2.381e-6**

of ground-truth $\boldsymbol{P}$ and $\boldsymbol{F}^{e,\text{new}}$, which are not easily accessible in the real world. Also, we train our method by back-propagation through time (BPTT). By contrast, labeled employs traditional training without time stepping. Our approach risks the training efficacy to some extent but is more natural to apply to training time-dependent physical systems. Finally, in continuum mechanics, different constitutive laws may describe the same kinematics (e.g., $\boldsymbol{F}$) with different stresses (e.g., $\boldsymbol{P}$). The goal of our method is to recover the kinematics ($\boldsymbol{x}$, $\boldsymbol{F}$) while being generalizable to new scenarios.

Despite not being as accurate as labeled when it comes to $\boldsymbol{P}$, our method predicts precise, generalizable kinematics. Our approach is also more stable than labeled across all environments, as shown in Tab. 3.1.

B.6.4 Data Efficiency

For the gnn, we keep as much as possible the training details in Sanchez-Gonzalez et al. [154]. Due to the loss of implementation details in their papers, we adapt the training procedure for spline and neural to be compatible with our method. In order to ablate the number of training trajectories to study how the amount of data affects the performance, we select gnn, which is the baseline with the strongest reliance on data, as the backbone and redo the training with 10 and 100 data trajectories. We present the loss comparison using the Water environment on the velocity generalization task, where we randomly sample 3 initial velocities and average the losses between the evaluation and the ground truth in Tab. B.4. The gradually decreasing loss with respect to the growing number of trajectories indicates gnn generalizes better with more data. However, even with 100x fewer data, ours still outperforms gnn by orders of magnitude, reflecting the efficiency of our method.

Table B.4: Data efficiency.

Method	#Trajectories	Loss
gnn	1	5.8e-2
gnn	10	2.5e-2
gnn	100	1.2e-2
ours	1	**1.9e-5**

Table B.5: Auxiliary loss.

Task	w/o vel loss	w/ vel loss
reconstruction	2.0e-5	**1.3e-5**
time	3.5e-4	**2.1e-4**
velocity	1.9e-5	**1.8e-5**
geometry	**2.4e-4**	3.0e-4

B.6.5 Auxiliary Loss

Here we study the impact of another important state variable: velocity. We additionally add a penalty on the dissimilarity between simulated velocities and the ground-truth velocities. We show the loss comparison of position using Water with or without velocity dissimilarity penalty in Tab. B.5. As indicated by the results, 3 out of 4 tasks slightly benefit from the velocity dissimilarity penalty. We argue that incorporating complementary losses could help the training to some extent, but it might be enough for a quick start to train with only positional supervision, which is usually the handiest ground truth in the real world.

B.6.6 More Visualization

- We visualize the training dataset in Fig. B-2.
- We compare the generalizability of our method with baselines and ground truth about different geometries in Figs. B-3 and B-4.

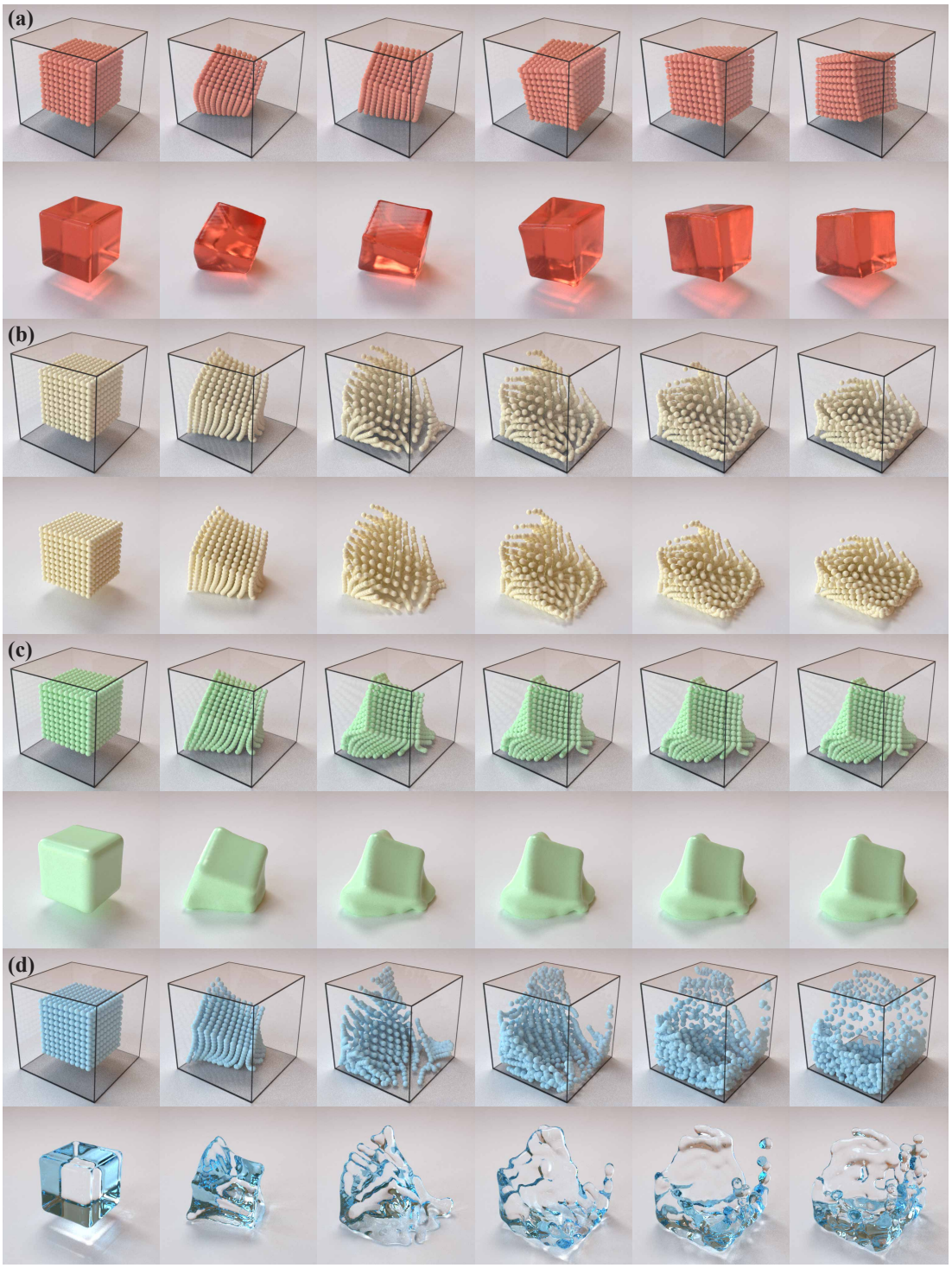


Figure B-2: Training Data. We show granular and realistic rendering for (a) Jell-O, (b) Sand, (c) Plasticine, and (d) Water.

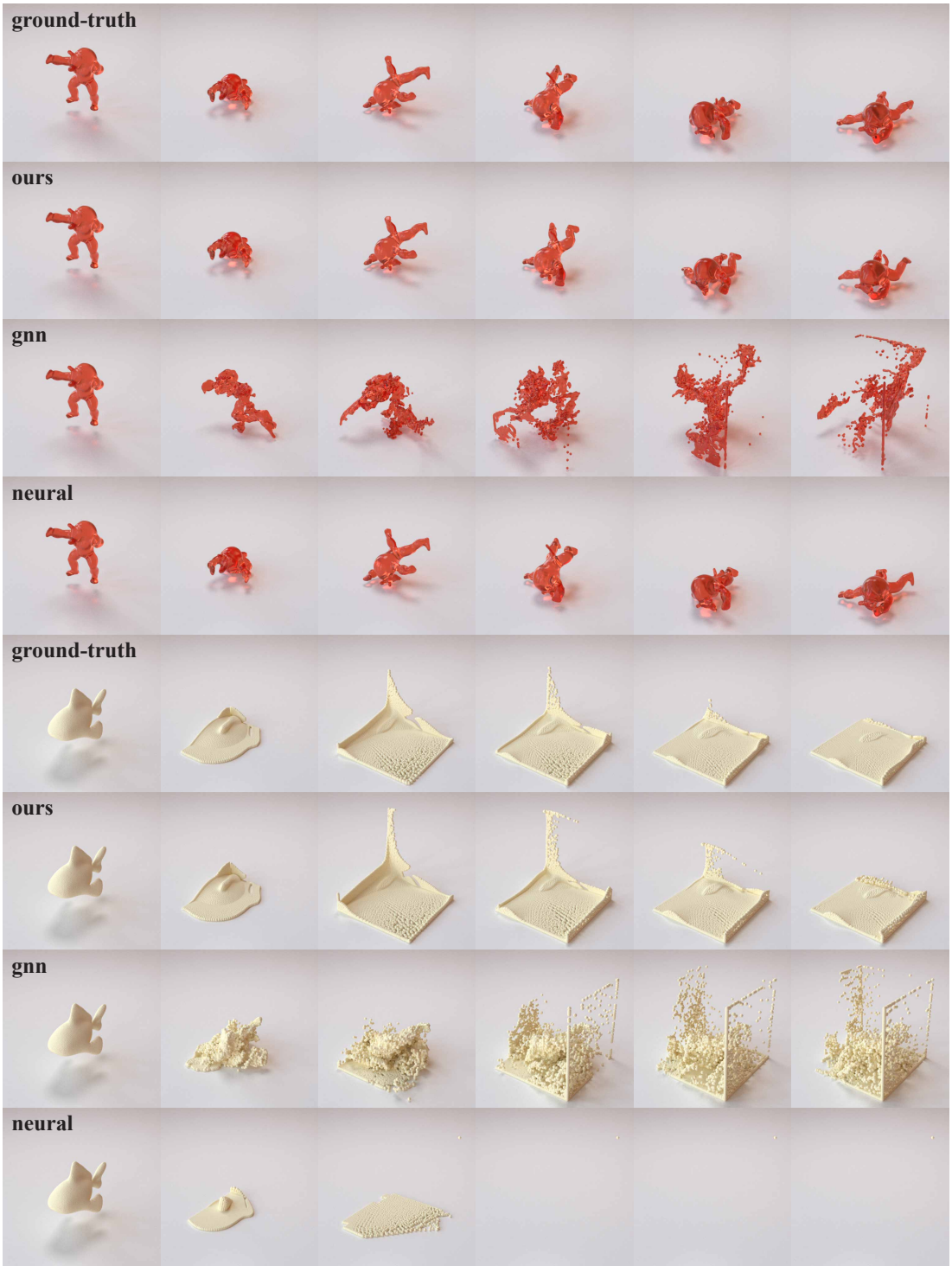


Figure B-3: Geometry Generalization for Jell-O and Sand. We set an armadillo geometry for Jell-O and a goldfish geometry for Sand. We compare ours with ground-truth and baselines including gnn and neural.

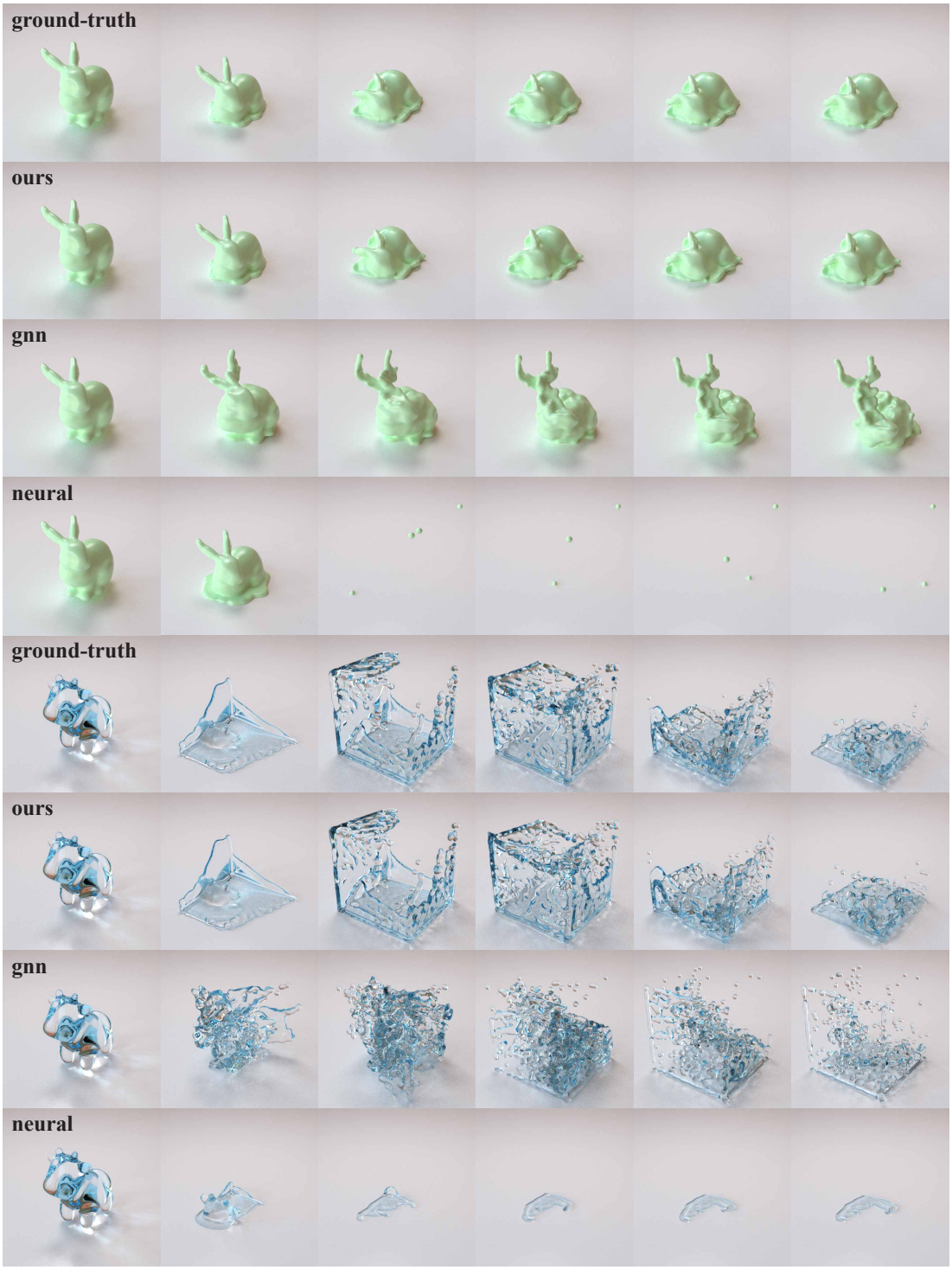


Figure B-4: Geometry Generalization for Plasticine and Water. We set a bunny geometry for Plasticine and a cow geometry for Water. We compare ours with ground-truth and baselines including gnn and neural.

Appendix C

Appendix for LLM and Simulation as Bilevel Optimizers: A New Paradigm to Advance Physical Scientific Discovery

C.1 Full Prompts

System prompt for constitutive law discovery:

```
You are an intelligent AI assistant for coding, physical simulation, and scientific discovery.
Follow the user's requirements carefully and make sure you understand them.
Your expertise is strictly limited to physical simulation, material science, mathematics, and coding.
Keep your answers short and to the point.
Do not provide any information that is not requested.
Always document your code as comments to explain the reason behind them.
Use Markdown to format your solution.
You are very familiar with Python and PyTorch.
Do not use any external libraries other than the libraries used in the examples.
```

System prompt for molecule design:

```
You are an intelligent AI assistant for coding, molecule design, and scientific discovery.
Follow the user's requirements carefully and make sure you understand them.
Your expertise is strictly limited to physical simulation, material science, chemistry, molecule design, mathematics, and
coding.
Keep your answers short and to the point.
Do not provide any information that is not requested.
Always document your code as comments to explain the reason behind them.
Use Markdown to format your solution.
You are very familiar with PyTorch.
Your are very familiar with the SMILES notation (Simplified Molecular-Input Line-Entry System).
Do not use any external libraries other than the libraries used in the examples.
```

Coding format prompt for elastic constitutive law discovery:

Format Requirements

PyTorch Tips
1. When element-wise multiplying two matrix, make sure their number of dimensions match before the operation. For example, when multiplying `J` (B,) and `I` (B, 3, 3), you should do `J.view(-1, 1, 1)` before the operation. Similarly, `(J - 1)` should also be reshaped to `(J - 1).view(-1, 1, 1)`. If you are not sure, write down every component in the expression one by one and annotate its dimension in the comment for verification.
2. When computing the trace of a tensor A (B, 3, 3), use `A.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1, 1)`. Avoid using `torch.trace` or `Tensor.trace` since they only support 2D matrix.

Code Requirements

1. The programming language is always python.
2. Annotate the size of the tensor as comment after each tensor operation. For example, `# (B, 3, 3)`.
3. The only library allowed is PyTorch. Follow the examples provided by the user and check the PyTorch documentation to learn how to use PyTorch.
4. Separate the code into continuous physical parameters that can be tuned with differentiable optimization and the symbolic constitutive law represented by PyTorch code. Define them respectively in the `__init__` function and the `forward` function.
5. The first output of the `forward` function is the updated deformation gradient. Always remember the second output of the `forward` function is Kirchhoff stress tensor, which is defined by the matrix multiplication between the first Piola-Kirchhoff stress tensor and the transpose of the deformation gradient tensor. Formally, `tau = P @ F^T`, where tau is the Kirchhoff stress tensor, P is the first Piola-Kirchhoff stress tensor, and F is the deformation gradient tensor. Do not directly return any other type of stress tensor other than Kirchhoff stress tensor. Compute Kirchhoff stress tensor using the equation: `tau = P @ F^T`.
6. The proposed code should strictly follow the structure and function signatures below:

```python
import torch
import torch.nn as nn


class Physics(nn.Module):

    def __init__(self, param: float = DEFAULT_VALUE):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        Tentatively initialize the parameters with the default values in args.

        Args:
            param (float): the physical meaning of the parameter.
        """
        super().__init__()
        self.param = nn.Parameter(torch.tensor(param))

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute Kirchhoff stress tensor from deformation gradient tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            kirchhoff_stress (torch.Tensor): Kirchhoff stress tensor (B, 3, 3).
        """
        return kirchhoff_stress
```

Solution Requirements

1. Analyze step-by-step what the potential problem is in the previous iterations based on the feedback. Think about why

```
the results from previous constitutive laws mismatched with the ground truth. Do not give advice about how to optimize. Focus on the formulation of the constitutive law. Start this section with "### Analysis". Analyze all iterations individually, and start the subsection for each iteration with "#### Iteration N", where N stands for the index. Remember to analyze every iteration in the history.
2. Think step-by-step what you need to do in this iteration. Think about how to separate your algorithm into a continuous physical parameter part and a symbolic constitutive law part. Describe your plan in pseudo-code, written out in great detail. Remember to update the default values of the trainable physical parameters based on previous optimizations. Start this section with "### Step-by-Step Plan".
3. Output the code in a single code block "```python ... ```" with detailed comments in the code block. Do not add any trailing comments before or after the code block. Start this section with "### Code".
```

Coding format prompt for plastic constitutive law discovery:

```
## Format Requirements

### PyTorch Tips
1. When element-wise multiplying two matrix, make sure their number of dimensions match before the operation. For example, when multiplying `J` (B,) and `I` (B, 3, 3), you should do `J.view(-1, 1, 1)` before the operation. Similarly, `(J - 1)` should also be reshaped to `(J - 1).view(-1, 1, 1)`. If you are not sure, write down every component in the expression one by one and annotate its dimension in the comment for verification.
2. When computing the trace of a tensor A (B, 3, 3), use `A.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1, 1)`. Avoid using `torch.trace` or `Tensor.trace` since they only support 2D matrix.

### Code Requirements

1. The programming language is always python.
2. Annotate the size of the tensor as comment after each tensor operation. For example, `# (B, 3, 3)`.
3. The only library allowed is PyTorch. Follow the examples provided by the user and check the PyTorch documentation to learn how to use PyTorch.
4. Separate the code into continuous physical parameters that can be tuned with differentiable optimization and the symbolic deformation gradient correction model represented by PyTorch code. Define them respectively in the `__init__` function and the `forward` function.
5. The proposed code should strictly follow the structure and function signatures below:

```python
import torch
import torch.nn as nn


class Physics(nn.Module):

    def __init__(self, param: float = DEFAULT_VALUE):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        Tentatively initialize the parameters with the default values in args.

        Args:
            param (float): the physical meaning of the parameter.
        """
        super().__init__()
        self.param = nn.Parameter(torch.tensor(param))

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute corrected deformation gradient from deformation gradient tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            F_corrected (torch.Tensor): corrected deformation gradient tensor (B, 3, 3).
```
```

```
      """
      return F_corrected
```

Solution Requirements

1. Analyze step-by-step what the potential problem is in the previous iterations based on the feedback. Think about why the results from previous constitutive laws mismatched with the ground truth. Do not give advice about how to optimize. Focus on the formulation of the constitutive law. Start this section with "### Analysis". Analyze all iterations individually, and start the subsection for each iteration with "#### Iteration N", where N stands for the index. Remember to analyze every iteration in the history.
2. Think step-by-step what you need to do in this iteration. Think about if the plasticity is needed to improve performance. Remember that plasticity is not necessary. If your analysis supports plasticity, think about how to update deformation gradient using plasticity. Think about how to separate your algorithm into a continuous physical parameter part and a symbolic deformation gradient correction model part. Describe your plan in pseudo-code, written out in great detail. Remember to update the default values of the trainable physical parameters based on previous optimizations. Start this section with "### Step-by-Step Plan".
3. Output the code in a single code block "```python ... ```" with detailed comments in the code block. Do not add any trailing comments before or after the code block. Start this section with "### Code".

Coding format prompt for molecule design:

```
## Format Requirements

### Code Requirements

1. The programming language is always python.
2. Annotate the size of the tensor as comment after each tensor operation. For example, `# (B, 3, 3)`.
3. Separate the code into: (1) python string `SMILES`: the SMILES string describing the molecular topology structure and atomic types, and (2) matrix `coordinates` the 3D coordinates of all atoms. These representations should not include hydrogens.
4. The SMILES string should be valid. Use your knowledge about Simplified Molecular-Input Line-Entry System to help you design a valid one.
5. The number of atoms in the SMILES string should be no less than 8, which means the number of atoms should be >= 8. Try to generate molecule with diverse atoms.
6. The 3D coordinates of the atoms should not be overlapping with each other. In another word, every row in the matrix `coordinates` should be distinct from each other.
7. The `coordinates` matrix is of shape `(N, 3)` where `N` stands for the number of atoms in the molecule. It should be identical to the number of atoms that the proposed SMILES string represents. State out the shape of any matrix defined in the comment as shown in the following example. State out the number of atoms that the SMILES string represents in the comment as shown in the following example.
8. The discrete SMILES string is critical in this problem since it defines the structure and cannot be tuned using differentiable optimization. Please propose different SMILES string from all examples or iterations above to discover and evaluate more structure. This is very important.
9. The proposed code should strictly follow the structure and function signatures below:

```python
SMILES: str # N atoms

coordinates: list[list[float]] # (N, 3)
```

Solution Requirements

1. Analyze step-by-step what the potential problem is in the previous iterations based on the feedback. Think about why the results from previous molecule structure mismatched with the ground truth. Do not give advice about how to optimize. Focus on the formulation of the SMILES string. Start this section with "### Analysis". Analyze all iterations individually, and start the subsection for each iteration with "#### Iteration N", where N stands for the index. Remember to analyze every iteration in the history.
2. Think step-by-step what you need to do in this iteration. Think about how to separate your algorithm into a continuous 3D coordinate system part and a discrete SMILES string part. Remember the SMILES string proposed should always be
```

```
different from previous iterations. After propose the new SMILES string, compute and count step-by-step how many atoms it
contains. The continuous parameter should follow the number of atoms in the SMILES string. Describe your plan in
pseudo-code, written out in great detail. Start this section with "### Step-by-Step Plan".
3. Output the code in a single code block "```python ... ```" with detailed comments in the code block. After the SMILES
string, compute the number of atoms in it by counting. Remember that the number of atoms in the SMILES string should be no
less than 8, which means the number of atoms should be >= 8. Try to generate molecule with diverse atoms. Do not add any
trailing comments before or after the code block. Start this section with "### Code".
```

C.2 More Explanations

C.2.1 Data Workflow

The full input to LLM has 3 main parts: (i) system prompt, (ii) iteration information, and (iii) format prompt. For the system prompt, we insert it into the LLM at the beginning or input it as a special instruction depending on the type of LLM. For the iteration information, we first concatenate the code and its feedback and then simply stack the top K solutions. Finally, we append the format prompt at the end of the prompt to regularize the expected output. From our experiments, it is important to keep the order of prompts to ensure the performance and the successful parsing. More precisely, we show this process in the following python-like code:

```python
prompts = []
prompts.append(system_prompt)
for solution in reversed(solutions.topk()):
    iteration_prompt = solution.code + '\n' + solution.feedback
    prompts.append(iteration_prompt)
prompts.append(format_prompt)
full_prompt = '\n'.join(prompts)
```

C.2.2 Differences to Symbolic Regression Task

- Our problem focuses on loss-guided general scientific discovery, which is a superset of regular regression problems. In the constitutive law search tasks, we do not directly feed the input/output pair to our method. Instead, we consider a much more challenging task: apply the generated constitutive law recursively and use the overall loss as the performance metric. Concretely, a classic SR methods solve $\arg\min_f \|f(X) - y\|$ given $< X, y >$ pairs, whereas our method solves $\arg\min_f \|g(f(X))\|$ given $< X, g(f(X)) >$ pairs and g is a complex function like

physical simulation. It is easy to construct g to cover the former case using the later formulation, proving the generality of our problem setup. We formulate our problem as such to reflect a more realistic scenario in scientific discovery, where direct supervision is extremely sparse.

- Our method supports arbitrary number of input variables and output features, where most of SR methods [179] have limitation on the number of input and output. The input limitation strongly caps the complexity of tasks they can solve, and the output limitation forces them ignore the structural correlation between each output dimension. In a comparison, our method supports arbitrary problem settings thanks to the code-based representation, which enables multi-dimensional arrays and tensor operations.

- Our model adapts to multi-discipline application easily, while traditional SR methods typically incorporate with domain-experts' priors via hard-coded constraints and heuristic [176], which is limited, domain-specific, and difficult to customize. Our method is built upon LLMs pre-trained on internet-level data that contains multi-discipline natural languages, mathematical expressions, and codes. As a result, it is easy for users to customize it and adapt to their own diverse applications via natural language guidance.

C.3 More Experiments

C.3.1 Symbolic Regression

We present the full results of the comparison to symbolic regression methods in Table C.1

C.3.2 Longer Iteration

In order to further investigate the potential of our method and ablate the hyper-parameters for practitioners, we add a new study in terms of the number of iterations (question-answering

Table C.1: Symbolic Regression

Method	R2 ↑	MSE ↓	MAE ↓	Symbolic
AIFeynman [176]	0.05105	22814675.8	2520.0	✓
DSR [136]	0.57527	10966411.0	2045.0	✓
BSR [78]	0.66526	8642965.0	1938.6	✓
AdaBoost [155]	0.75058	6439962.9	1777.7	✗
GP-GOMEA [182]	0.77734	5749076.4	1580.1	✓
SBP-GP [181]	0.81773	4706077.0	1367.5	✓
LightGBM [82]	0.83368	4294433.7	1129.9	✗
XGBoost [24]	0.87775	3156500.5	1109.2	✗
MRGP [6]	0.91074	2304682.5	950.5	✓
EPLEX [90]	0.91851	2104070.1	122.2	✓
FFX [120]	0.93124	1775263.7	801.7	✓
MLP	0.98240	454461.5	366.3	✗
FEAT [21]	0.98761	319800.6	336.1	✓
DSO [125]	0.99642	92374.9	168.6	✓
Operon [88]	0.99684	81577.9	92.4	✓
SymbolicGPT [179]	0.52333	6862154.7	1680.7	✓
NeSymReS [13]	N/A to >3 variables			✓
T-JSL [99]	N/A to >2 variables			✓
Ours	**0.99901**	**17424.6**	**86.4**	✓

Table C.2: Longer Iteration

#Iterations	(a) ↓	(b) ↓	(c) ↓	(d) ↓	(e) ↓	(f) ↓	(g) ↓	(h) ↓
5	5.2e-5	2.1e-1	6.0e-2	**1.4e-12**	**1.3e-4**	1.1e-1	5.4e-1	3.6e-5
20	**4.2e-6**	**4.0e-4**	**2.5e-3**	**1.4e-12**	**1.3e-4**	**6.5e-2**	**1.2e-1**	**5.6e-6**
Improvement	+1138.1%	+52400.0%	+2300.0%	0.0%	0.0%	+69.2%	+350.0%	+542.9%

cycles). We repeat our experiment in Table 4.1 with a prolonged number of iterations to 20 and report the performance in Table C.2.

As shown in the table, the number of iterations turns out to be a determining hyper-parameter with significant impart on the performance. While it has little affect on relatively easier tasks, it dramatically improves the performance of the most challenging tasks including **(b)** and **(c)**. For practitioners, the number of iteration should be first considered as the most important hyper-parameter when adapting our method to their own tasks.

C.4 More Results

C.4.1 Constitutive Law Discovery (a)

The best solution on task **(a)** optimized by our method:

```python
import torch
import torch.nn as nn

class Physics(nn.Module):
    # Best values from the training curves
    DEFAULT_YOUNGS_MODULUS_LOG = 13.03
    DEFAULT_POISSONS_RATIO_SIGMOID = -1.99

    def __init__(self, youngs_modulus_log: float = DEFAULT_YOUNGS_MODULUS_LOG, poissons_ratio_sigmoid: float = DEFAULT_POISSONS_RATIO_SIGMOID):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        Initialize the parameters with the best values from previous feedback.
        """
        super().__init__()
        # Initialize the parameters as trainable parameters
        self.youngs_modulus_log = nn.Parameter(torch.tensor(youngs_modulus_log))  # Log of Young's modulus
        self.poissons_ratio_sigmoid = nn.Parameter(torch.tensor(poissons_ratio_sigmoid))  # Sigmoid of Poisson's ratio

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute Kirchhoff stress tensor from deformation gradient tensor.

        Args:
            F (torch.Tensor): Deformation gradient tensor (B, 3, 3).

        Returns:
            kirchhoff_stress (torch.Tensor): Kirchhoff stress tensor (B, 3, 3).
        """
        # Convert the parameters to their actual values
        youngs_modulus = self.youngs_modulus_log.exp()  # (1,)
        poissons_ratio = torch.sigmoid(self.poissons_ratio_sigmoid) * 0.49  # (1,)

        # Lame parameters
        mu = youngs_modulus / (2 * (1 + poissons_ratio))  # Shear modulus (1,)
        lam = youngs_modulus * poissons_ratio / ((1 + poissons_ratio) * (1 - 2 * poissons_ratio))  # First Lame parameter (1,)

        # Deformation gradient determinant J and its reshape for operations (B,)
        J = F.det().view(-1, 1, 1)

        # Inverse transpose of F for stress computation (B, 3, 3)
        F_invT = F.inverse().transpose(1, 2)

        # Compute first Piola-Kirchhoff stress tensor P (B, 3, 3)
        # Volumetric part
        P_vol = lam * (J - 1) * F_invT

        # Deviatoric part combining neo-Hookean behavior
        # This accounts for the near incompressible nature of the material
        P_dev = mu * (F - (1 / J) * F_invT)

```

```
        # Compute Kirchhoff stress tensor tau by multiplying the first Piola-Kirchhoff with the transpose of F (B, 3, 3)
        kirchhoff_stress = P_vol + P_dev @ F.transpose(1, 2)

        return kirchhoff_stress
```

C.4.2 Constitutive Law Discovery (b)

The best solution on task **(b)** optimized by our method:

```
import torch
import torch.nn as nn

class Physics(nn.Module):

    def __init__(self, gamma: float = -0.07):  # Based on best value from iteration 5
        """
        Initialize gamma as a trainable parameter which will be used for scaling the soft
        deformation correction.

        Args:
            gamma (float): scaling factor for the deformation correction.
        """
        super().__init__()
        self.gamma = nn.Parameter(torch.tensor(gamma))  # Initialize gamma, (1,)

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute corrected deformation gradient tensor F, by applying a soft correction
        proportional to the deviation of its determinant from 1, effectively guiding the
        gradient towards physically realistic states.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            F_corrected (torch.Tensor): corrected deformation gradient tensor (B, 3, 3).
        """
        # Compute determinant of F and create a condition based on its value, (B,)
        J = torch.det(F)  # (B,)

        # Apply a smooth step function as a deviation condition, (B,)
        J_deviation_condition = torch.tanh(J - 1)  # (B,)

        # Prepare for correction, taking into account the batch dimension (B,)
        gamma_correction = self.gamma * J_deviation_condition.view(-1, 1, 1)  # (B, 1, 1)

        # Identity matrix, expanded for batch size (B, 3, 3)
        I = torch.eye(3, device=F.device).repeat(F.size(0), 1, 1)  # (B, 3, 3)

        # Correct F by pulling towards identity matrix when determinant deviates from 1, (B, 3, 3)
        F_corrected = F - gamma_correction * (F - I)  # (B, 3, 3)

        return F_corrected
```

C.4.3 Constitutive Law Discovery (c)

The best solution on task **(c)** optimized by our method:

```python
import torch
import torch.nn as nn

# The default value for elastic_limit is set to the best from the last iteration, and
# we initialize a new parameter for capturing the hardening effect
DEFAULT_ELASTIC_LIMIT = 0.92
DEFAULT_HARDENING_FACTOR = 0.1

class Physics(nn.Module):

    def __init__(self, elastic_limit: float = DEFAULT_ELASTIC_LIMIT,
                 hardening_factor: float = DEFAULT_HARDENING_FACTOR):
        """
        Define trainable continuous physical parameters for differentiable optimization.

        Args:
            elastic_limit (float): the parameter determining the initial yield strength.
            hardening_factor (float): the parameter controlling the rate of hardening.
        """
        super().__init__()
        self.elastic_limit = nn.Parameter(torch.tensor(elastic_limit))  # ()
        self.hardening_factor = nn.Parameter(torch.tensor(hardening_factor))  # ()

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute corrected deformation gradient from deformation gradient tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            F_corrected (torch.Tensor): corrected deformation gradient tensor (B, 3, 3).
        """
        # Obtain the polar decomposed rotation (R) and stretch (S)
        U, S, V = torch.svd(F)  # U: (B, 3, 3), S: (B, 3), V: (B, 3, 3)
        R = U @ V.transpose(-2, -1)  # R: (B, 3, 3)

        # Correct the S tensor with hardening
        # Assuming hardening affects the elastic limit linearly with accumulated plastic strain
        plastic_strain = torch.relu(S - self.elastic_limit)  # Presumed plastic strain
        hardening_adjustment = 1.0 + (self.hardening_factor * plastic_strain)

        S_clamped = torch.min(S, self.elastic_limit * hardening_adjustment)  # Clamp S with hardening
        S_corrected = torch.diag_embed(S_clamped)  # S_corrected: (B, 3, 3)

        F_corrected = R @ S_corrected  # (B, 3, 3) Corrected deformation gradient tensor

        # Ensure volume preservation
        J = torch.det(F).view(-1, 1, 1)  # (B, 1, 1) Determinant of the input F for volume
        J_corrected = torch.det(F_corrected).view(-1, 1, 1)  # (B, 1, 1) Determinant of the corrected F
        volume_ratio = (J / J_corrected) ** (1/3)
        F_corrected = F_corrected * volume_ratio  # (B, 3, 3) Volume-preserved F_corrected

        return F_corrected
```

C.4.4 Constitutive Law Discovery (d)

The best solution on task **(d)** optimized by our method:

```python
import torch
import torch.nn as nn

class Physics(nn.Module):

    DEFAULT_VALUE = 0.0  # Best guess based on previous behavior

    def __init__(self, param: float = DEFAULT_VALUE):
        """
        Define trainable continuous physical parameters for differentiable optimization.
        The parameter modulates corrections towards nearly isochoric behavior.
        """
        super().__init__()
        self.param = nn.Parameter(torch.tensor([param]))  # Scalar modulation parameter

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Symbolic deformation gradient correction model.
        """
        # Compute the determinant of the deformation gradient (volumetric change)
        J = torch.det(F)  # (B,)

        # Compute the volumetric part of the deformation gradient: J^(1/3)*I
        I = torch.eye(3).to(F.device)  # (3, 3)
        # Expand the identity matrix to the entire batch
        I = I.view(1, 3, 3).expand(F.size(0), -1, -1)  # (B, 3, 3)
        # Calculate volumetric part
        vol_deform = torch.pow(J, 1.0 / 3.0).view(-1, 1, 1) * I  # (B, 3, 3)

        # Calculate the deviatoric part of F: divide F by J^(1/3)
        dev_deform = F / torch.pow(J, 1.0 / 3.0).view(-1, 1, 1)  # (B, 3, 3)

        # Modulate correction by self.param and construct the correction term
        correction = self.param * (I - dev_deform)  # (B, 3, 3)

        # Combine the volumetric part with the deviatoric correction
        F_corrected = vol_deform + correction  # (B, 3, 3)

        return F_corrected  # (B, 3, 3)
```

C.4.5 Imaginary Constitutive Law

The best solution on the imaginary constitutive law invention task optimized by our method:

```python
import torch
import torch.nn as nn

# Default values for the physical parameters based on previous iterations
DEFAULT_KAPPA = 0.08
DEFAULT_MU = 0.28

class Physics(nn.Module):
    def __init__(self, kappa: float = DEFAULT_KAPPA, mu: float = DEFAULT_MU):
        """
```

```
        Initialize the continuous physical parameters kappa and mu for differentiable optimization.
        """
        super().__init__()
        self.kappa = nn.Parameter(torch.tensor(kappa))  # Bulk modulus correction factor (scalar)
        self.mu = nn.Parameter(torch.tensor(mu))       # Shear modulus correction factor (scalar)

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute the corrected deformation gradient from the deformation gradient tensor.
        """
        B = F.size(0)  # Batch size (scalar)
        I = torch.eye(3, device=F.device).unsqueeze(0).expand(B, -1, -1)  # Identity matrix (B, 3, 3)
        J = torch.det(F).view(-1, 1, 1)  # Jacobian determinant (B, 1, 1)

        # Volume correction factor (B, 1, 1)
        vol_correction_factor = torch.clamp(self.kappa * (J - 1), min=0.0, max=1.0)
        vol_correction = vol_correction_factor * I  # Volume correction term (B, 3, 3)

        # Compute trace of F for shape correction (B, 1, 1)
        trace_F = F.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1, 1)
        dev_F = F - (trace_F / 3) * I  # Deviatoric part of F (B, 3, 3)

        # Shape correction factor (scalar)
        shape_correction_factor = torch.clamp(self.mu, min=0.0, max=1.0)
        shape_correction = shape_correction_factor * dev_F  # Shape correction term (B, 3, 3)

        F_corrected = F - vol_correction - shape_correction  # Corrected deformation gradient (B, 3, 3)
        return F_corrected
```

Bibliography

[1] S Mazdak Abulnaga, Esra Abaci Turk, Mikhail Bessmeltsev, P Ellen Grant, Justin Solomon, and Polina Golland. Placental flattening via volumetric parameterization. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 39–47. Springer, 2019.

[2] Microsoft Research AI4Science and Microsoft Azure Quantum. The impact of large language models on scientific discovery: a preliminary study using gpt-4. *arXiv preprint arXiv:2311.07361*, 2023.

[3] Muhammad Rizwan Amjid, Aamir Shehzad, Shahzad Hussain, Muhammad Asim Shabbir, Moaazam Rafiq Khan, Muhammad Shoaib, et al. A comprehensive review on wheat flour dough rheology. *Pakistan Journal of Food Sciences*, 23(2):105–123, 2013.

[4] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[5] Anthropic. Introducing the next generation of claude, 2024. URL https://www.anthropic.com/news/claude-3-family.

[6] Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 879–886, 2014.

[7] Ellen M Arruda and Mary C Boyce. A three-dimensional constitutive model for the large stretch behavior of rubber elastic materials. *Journal of the Mechanics and Physics of Solids*, 41(2):389–412, 1993.

[8] Faisal As' ad, Philip Avery, and Charbel Farhat. A mechanics-informed artificial neural network approach in data-driven constitutive modeling. *International Journal for Numerical Methods in Engineering*, 123(12):2738–2759, 2022.

[9] Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

[10] 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, 2019.

[11] Jernej Barbič and Doug L James. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM transactions on graphics (TOG)*, 24(3):982–990, 2005.

[12] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR, 2018.

[13] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pages 936–945. Pmlr, 2021.

[14] Alan Wilfred Bishop and David John Henkel. The measurement of soil properties in the triaxial test. 1962.

[15] Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

[16] Ronaldo I Borja. *Plasticity*, volume 2. Springer, 2013.

[17] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024. URL https://openai.com/research/video-generation-models-as-world-simulators.

[18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[19] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[20] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *International Conference on Learning Representations*, 2024.

[21] William La Cava, Tilak Raj Singh, James Taggart, Srinivas Suri, and Jason Moore. Learning concise representations for regression by evolving networks of trees. In *International Conference on Learning Representations*, 2019.

[22] Hsiao-yu Chen, Edith Tretschk, Tuur Stuyck, Petr Kadlecek, Ladislav Kavan, Etienne Vouga, and Christoph Lassner. Virtual elastic objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15827–15837, 2022.

[23] Peter Yichen Chen, Maytee Chantharayukhonthorn, Yonghao Yue, Eitan Grinspun, and Ken Kamrin. Hybrid discrete-continuum modeling of shear localization in granular media. *Journal of the Mechanics and Physics of Solids*, 153:104404, 2021.

[24] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[25] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[26] Raj P Chhabra. *Bubbles, drops, and particles in non-Newtonian fluids*. CRC press, 2006.

[27] Seyone Chithrananda, Gabriel Grand, and Bharath Ramsundar. Chemberta: large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885*, 2020.

[28] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153:235–256, 2007.

[29] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *The Twelfth International Conference on Learning Representations*.

[30] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in Neural Information Processing Systems*, 31, 2018.

[31] Eduardo A de Souza Neto, Djordje Peric, and David RJ Owen. *Computational methods for plasticity: theory and applications*. John Wiley & Sons, 2011.

[32] Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, page 6, 2019.

[33] Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas J Guibas. Vector neurons: A general framework for so (3)-equivariant networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12200–12209, 2021.

[34] Daniel Charles Drucker and William Prager. Soil mechanics and plastic analysis or limit design. *Quarterly of applied mathematics*, 10(2):157–165, 1952.

[35] Tao Du, Kui Wu, Andrew Spielberg, Wojciech Matusik, Bo Zhu, and Eftychios Sifakis. Functional optimization of fluidic devices with differentiable stokes flow. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[36] Tao Du, Josie Hughes, Sebastien Wah, Wojciech Matusik, and Daniela Rus. Underwater soft robot modeling and control with differentiable simulation. *IEEE Robotics and Automation Letters*, 6(3):4994–5001, 2021.

[37] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics (TOG)*, 41(2):1–21, 2021.

[38] GW Ellis, C Yao, Rui Zhao, and Df Penumadu. Stress-strain modeling of sands using artificial neural networks. *Journal of geotechnical engineering*, 121(5):429–435, 1995.

[39] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

[40] Xiaomin Fang, Lihang Liu, Jieqiong Lei, Donglong He, Shanzhuo Zhang, Jingbo Zhou, Fan Wang, Hua Wu, and Haifeng Wang. Geometry-enhanced molecular representation learning for property prediction. *Nature Machine Intelligence*, 4(2): 127–134, 2022.

[41] Santo Fortunato, Carl T Bergstrom, Katy Börner, James A Evans, Dirk Helbing, Staša Milojević, Alexander M Petersen, Filippo Radicchi, Roberta Sinatra, Brian Uzzi, et al. Science of science. *Science*, 359(6379):eaao0185, 2018.

[42] Alexander Fuchs, Yousef Heider, Kun Wang, WaiChing Sun, and Michael Kaliske. Dnn2: A hyper-parameter reinforcement learning game for self-design of neural network based elasto-plastic constitutive descriptions. *Computers & Structures*, 249: 106505, 2021.

[43] YC Fung. Elasticity of soft tissues in simple elongation. *American Journal of Physiology-Legacy Content*, 213(6):1532–1544, 1967.

[44] Tomonari Furukawa and Genki Yagawa. Implicit constitutive modelling for viscoplasticity using neural networks. *International Journal for Numerical Methods in Engineering*, 43(2):195–219, 1998.

[45] Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, et al. ThreeDWorld: A platform for interactive multi-modal physical simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[46] Ming Gao, Xinlei Wang, Kui Wu, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. Gpu optimization of material point methods. *ACM Transactions on Graphics (TOG)*, 37(6):1–12, 2018.

[47] 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 Trans. Graph.*, 39(6), nov 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417766. URL https://doi.org/10.1145/3414685.3417766.

[48] Jamshid Ghaboussi, JH Garrett Jr, and Xiping Wu. Knowledge-based modeling of material behavior with neural networks. *Journal of engineering mechanics*, 117(1): 132–153, 1991.

[49] Oscar Gonzalez and Andrew M Stuart. *A first course in continuum mechanics*, volume 42. Cambridge University Press, 2008.

[50] Morton E Gurtin, Eliot Fried, and Lallit Anand. *The mechanics and thermodynamics of continua*. Cambridge University Press, 2010.

[51] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

[52] Richard Haberman. *Mathematical models: mechanical vibrations, population dynamics, and traffic flow*. SIAM, 1998.

[53] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[54] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[55] David Hahn, Pol Banzet, James M. Bern, and Stelian Coros. Real2Sim: Visco-elastic parameter estimation from dynamic motion. *ACM Trans. Graph.*, 38(6), nov 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356548. URL https://doi.org/10.1145/3355089.3356548.

[56] Thomas A Halgren. Merck molecular force field. i. basis, form, scope, parameterization, and performance of mmff94. *Journal of computational chemistry*, 17(5-6): 490–519, 1996.

[57] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

[58] Nicklas Hansen, Jyothir SV, Vlad Sobal, Yann LeCun, Xiaolong Wang, and Hao Su. Hierarchical world models as visual whole-body humanoid controllers. *arXiv preprint arXiv:2405.18418*, 2024.

[59] Nikolaus Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pages 75–102, 2006.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[61] H Hencky. The elastic behavior of vulcanized rubber. *Rubber Chemistry and Technology*, 6(2):217–224, 1933.

[62] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[63] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[64] Yining Hong, Li Yi, Joshua B Tenenbaum, Antonio Torralba, and Chuang Gan. PTR: A benchmark for part-based conceptual, relational, and physical reasoning. In *Advances In Neural Information Processing Systems*, 2021.

[65] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[66] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[67] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Fredo Durand. Difftaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2020.

[68] Daniel Z Huang, Kailai Xu, Charbel Farhat, and Eric Darve. Learning constitutive relations from indirect observations using deep neural networks. *Journal of Computational Physics*, 416:109491, 2020.

[69] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Benchmarking large language models as ai research agents. *arXiv preprint arXiv:2310.03302*, 2023.

[70] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B Tenenbaum, and Chuang Gan. PlasticineLab: A soft-body manipulation benchmark with differentiable physics. In *International Conference on Learning Representations*, 2020.

[71] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[72] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12619–12629, 2019. doi: 10.1109/CVPR.2019.01291.

[73] Wojciech Jarosz, Volker Schönefeld, Leif Kobbelt, and Henrik Wann Jensen. Theory, analysis and applications of 2d global illumination. 31(5), sep 2012. ISSN 0730-0301. doi: 10.1145/2231816.2231823. URL https://doi.org/10.1145/2231816.2231823.

[74] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[75] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):1–10, 2015.

[76] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, pages 1–52. 2016.

[77] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018.

[78] Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

[79] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[80] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2019.

[81] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6): 422–440, 2021.

[82] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[83] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4): 139–1, 2023.

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

[85] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[86] Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. Drucker-prager elastoplasticity for sand animation. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[87] Dominik K Klein, Mauricio Fernández, Robert J Martin, Patrizio Neff, and Oliver Weeger. Polyconvex anisotropic hyperelasticity with neural networks. *Journal of the Mechanics and Physics of Solids*, 159:104703, 2022.

[88] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, 21(3):471–501, 2020.

[89] Stefan Kramer, Mattia Cerrato, Sašo Džeroski, and Ross King. Automated scientific discovery: From equation discovery to autonomous discovery systems. *arXiv preprint arXiv:2305.02251*, 2023.

[90] William La Cava, Thomas Helmuth, Lee Spector, and Jason H Moore. A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. *Evolutionary Computation*, 27(3):377–402, 2019.

[91] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason Moore. Contemporary symbolic regression methods and their relative performance. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[92] Lev Davidovich Landau, JS Bell, MJ Kearsley, LP Pitaevskii, EM Lifshitz, and JB Sykes. *Electrodynamics of continuous media*, volume 8. elsevier, 2013.

[93] Greg Landrum et al. Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling. *Greg Landrum*, 8:31, 2013.

[94] BA Le, Julien Yvonnet, and Q-C He. Computational homogenization of nonlinear elastic materials using neural networks. *International Journal for Numerical Methods in Engineering*, 104(12):1061–1084, 2015.

[95] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022.

[96] Jiatong Li, Yunqing Liu, Wenqi Fan, Xiao-Yong Wei, Hui Liu, Jiliang Tang, and Qing Li. Empowering molecule discovery for molecule-caption translation with large language models: A chatgpt perspective. *arXiv preprint arXiv:2306.06615*, 2023.

[97] Tzu-Mao Li, Jaakko Lehtinen, Ravi Ramamoorthi, Wenzel Jakob, and Frédo Durand. Anisotropic gaussian mutations for metropolis light transport through hessian-hamiltonian dynamics. *ACM Trans. Graph.*, 34(6), oct 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818084. URL https://doi.org/10.1145/2816795.2818084.

[98] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6), dec 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275109. URL https://doi.org/10.1145/3272127.3275109.

[99] Wenqiang Li, Weijun Li, Linjun Sun, Min Wu, Lina Yu, Jingyi Liu, Yanjie Li, and Songsong Tian. Transformer-based model for symbolic regression via joint supervised learning. In *The Eleventh International Conference on Learning Representations*, 2022.

[100] Xuan Li, Yadi Cao, Minchen Li, Yin Yang, Craig Schroeder, and Chenfanfu Jiang. Plasticitynet: Learning to simulate metal, sand, and snow for optimization time integration. *Advances in Neural Information Processing Systems*, 35:27783–27796, 2022.

[101] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2018.

[102] Zijie Li and Amir Barati Farimani. Graph neural network-accelerated lagrangian fluid simulation. *Computers & Graphics*, 103:201–211, 2022.

[103] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[104] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 32, 2019.

[105] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[106] Burigede Liu, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Andrew M Stuart, and Kaushik Bhattacharya. A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids*, 158:104668, 2022.

[107] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[108] Minliang Liu, Liang Liang, and Wei Sun. A generic physics-informed neural network-based constitutive model for soft biological tissues. *Computer methods in applied mechanics and engineering*, 372:113402, 2020.

[109] Risheng Liu, Pan Mu, Xiaoming Yuan, Shangzhi Zeng, and Jin Zhang. A general descent aggregation framework for gradient-based bi-level optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):38–57, 2022.

[110] Zhichao Liu, Ruth A Roberts, Madhu Lal-Nag, Xi Chen, Ruili Huang, and Weida Tong. Ai-based language models powering drug discovery and development. *Drug Discovery Today*, 26(11):2593–2607, 2021.

[111] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[112] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[113] Pingchuan Ma, Tao Du, Joshua B Tenenbaum, Wojciech Matusik, and Chuang Gan. RISP: Rendering-invariant state predictor with differentiable simulation and rendering for cross-domain parameter estimation. In *International Conference on Learning Representations*, 2021.

[114] 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):1–14, 2021.

[115] 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.

[116] Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *International Conference on Machine Learning*. PMLR, 2024.

[117] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

[118] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

[119] Agata Marzec, Jolanta Kowalska, Ewa Domian, Sabina Galus, Agnieszka Ciurzyńska, and Hanna Kowalska. Characteristics of dough rheology and the structural, mechanical, and sensory properties of sponge cakes with sweeteners. *Molecules*, 26(21):6638, 2021.

[120] Trent McConaghy. Ffx: Fast, scalable, deterministic symbolic regression technology. *Genetic Programming Theory and Practice IX*, pages 235–260, 2011.

[121] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, page 449–456, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 9781450378239. doi: 10.1145/1186562.1015744. URL https://doi.org/10.1145/1186562.1015744.

[122] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[123] R v Mises. Mechanik der festen körper im plastisch-deformablen zustand. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1913:582–592, 1913.

[124] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30:543–574, 1992.

[125] T. Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P. Santiago, Daniel M. Faissol, and Brenden K. Petersen. Symbolic regression via neural-guided genetic programming population seeding. In *Advances in Neural Information Processing Systems*, 2021.

[126] J Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, et al. gradsim: Differentiable simulation for system identification and visuomotor control. In *International Conference on Learning Representations*, 2020.

[127] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. Graph.*, 38(6), nov 2019. ISSN 0730-0301. doi: 10.1145/3355089.3356498. URL https://doi.org/10.1145/3355089.3356498.

[128] Raymond W Ogden. *Non-linear elastic deformations*. Courier Corporation, 1997.

[129] OpenAI. OpenAI: Introducing ChatGPT, 2022. URL https://openai.com/blog/chatgpt.

[130] OpenAI. OpenAI: GPT-4, 2023. URL https://openai.com/research/gpt-4.

[131] OptiTrack. OptiTrack motion capture systems. https://optitrack.com/. Accessed: 2021-10-05.

[132] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[133] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[134] 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. *Advances in neural information processing systems*, 32, 2019.

[135] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810, 2018. doi: 10.1109/ICRA.2018.8460528.

[136] Brenden K Petersen, Mikel Landajuela Larma, Terrell N Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2020.

[137] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020.

[138] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016.

[139] Karl Popper. *The logic of scientific discovery*. Routledge, 2005.

[140] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Scalable differentiable physics for learning and control. In *International Conference on Machine Learning*, pages 7847–7856. PMLR, 2020.

[141] 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.

[142] Yiling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Differentiable simulation of soft multi-body systems. *Advances in Neural Information Processing Systems*, 34: 17123–17135, 2021.

[143] Qualisys. Qualisys motion capture systems. https://www.qualisys.com/. Accessed: 2021-10-05.

[144] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[145] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017.

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

[147] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[148] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[149] Ravi Ramamoorthi, Dhruv Mahajan, and Peter Belhumeur. A first-order analysis of lighting, shading, and shadows. 26(1):2–es, jan 2007. ISSN 0730-0301. doi: 10.1145/1189762.1189764. URL https://doi.org/10.1145/1189762.1189764.

[150] Sereina Riniker and Gregory A Landrum. Better informed distance geometry: using what we know to improve conformation generation. *Journal of chemical information and modeling*, 55(12):2562–2574, 2015.

[151] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, pages 1–3, 2023.

[152] Alex Rosenberg and Lee McIntyre. *Philosophy of science: A contemporary introduction*. Routledge, 2019.

[153] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. In *Robotics: Science and Systems*, 2017.

[154] 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.

[155] Robert E Schapire. The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, pages 149–171, 2003.

[156] Gisbert Schneider. Automating drug discovery. *Nature reviews drug discovery*, 17 (2):97–113, 2018.

[157] Gaurav Sharma and Abhishek Thakur. Chatgpt in drug discovery. 2023.

[158] Yuelin Shen, K Chandrashekhara, WF Breig, and LR Oliver. Finite element analysis of v-ribbed belts using neural network based hyperelastic material model. *International Journal of Non-Linear Mechanics*, 40(6):875–890, 2005.

[159] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. Evolutionary algorithm for bilevel optimization using approximations of the lower level optimal solution mapping. *European Journal of Operational Research*, 257(2):395–411, 2017.

[160] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017.

[161] Alexey Stomakhin, Russell Howes, Craig A Schroeder, and Joseph M Teran. Energetically consistent invertible elasticity. In *Symposium on Computer Animation*, volume 1, 2012.

[162] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

[163] Alexey Stomakhin, Craig Schroeder, Chenfanfu Jiang, Lawrence Chai, Joseph Teran, and Andrew Selle. Augmented mpm for phase-change and varied materials. *ACM Transactions on Graphics (TOG)*, 33(4):1–11, 2014.

[164] Deborah Sulsky, Shi-Jian Zhou, and Howard L Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer physics communications*, 87(1-2): 236–252, 1995.

[165] Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. Survey Certification.

[166] Xiao Sun, Bahador Bahmani, Nikolaos N Vlassis, WaiChing Sun, and Yanxun Xu. Data-driven discovery of interpretable causal relations for deep learning material laws with uncertainty propagation. *Granular Matter*, 24(1):1–32, 2022.

[167] Andre Pradhana Tampubolon, Theodore Gast, Gergely Klár, Chuyuan Fu, Joseph Teran, Chenfanfu Jiang, and Ken Museth. Multi-species simulation of porous sand and water mixtures. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[168] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*, 2018.

[169] Alexandre M Tartakovsky, Carlos Ortiz Marrero, Paris Perdikaris, Guzel D Tartakovsky, and David Barajas-Solano. Learning parameters and constitutive relationships with physics informed deep neural networks. *arXiv preprint arXiv:1808.03398*, 2018.

[170] Joseph O Thompson. Hooke's law. *Science*, 64(1656):298–299, 1926.

[171] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. doi: 10.1109/IROS.2017.8202133.

[172] LRG Treloar. The elasticity of a network of long-chain molecules. i. *Transactions of the Faraday Society*, 39:36–41, 1943.

[173] Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[174] Cameron Tropea, Alexander L Yarin, John F Foss, et al. *Springer handbook of experimental fluid mechanics*, volume 1. Springer, 2007.

[175] Clifford Truesdell and Walter Noll. The non-linear field theories of mechanics. In *The non-linear field theories of mechanics*, pages 1–579. Springer, 2004.

[176] Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *Advances in Neural Information Processing Systems*, 33:4860–4871, 2020.

[177] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

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

[179] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131*, 2021.

[180] Vicon. VICON: Award-winning motion capture systems. https://www.vicon.com/. Accessed: 2021-10-05.

[181] Marco Virgolin, Tanja Alderliesten, and Peter AN Bosman. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pages 1084–1092, 2019.

[182] Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter AN Bosman. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation*, 29(2):211–237, 2021.

[183] Nikolaos N Vlassis and WaiChing Sun. Sobolev training of thermodynamic-informed neural networks for interpretable elasto-plasticity models with level set hardening. *Computer Methods in Applied Mechanics and Engineering*, 377:113695, 2021.

[184] Nikolaos N Vlassis and WaiChing Sun. Component-based machine learning paradigm for discovering rate-dependent and pressure-sensitive level-set plasticity models. *Journal of Applied Mechanics*, 89(2), 2022.

[185] Nikolaos N Vlassis and WaiChing Sun. Geometric deep learning for computational mechanics part ii: Graph embedding for interpretable multiscale plasticity. *arXiv preprint arXiv:2208.00246*, 2022.

[186] Nikolaos N Vlassis, Ran Ma, and WaiChing Sun. Geometric deep learning for computational mechanics part i: anisotropic hyperelasticity. *Computer Methods in Applied Mechanics and Engineering*, 371:113299, 2020.

[187] Nikolaos N Vlassis, Puhan Zhao, Ran Ma, Tommy Sewell, and WaiChing Sun. Molecular dynamics inferred transfer learning models for finite-strain hyperelasticity of monoclinic crystals: Sobolev training and validations against physical constraints. *International Journal for Numerical Methods in Engineering*, 2022.

[188] Bin Wang, Yuanmin Deng, Paul Kry, Uri Ascher, Hui Huang, and Baoquan Chen. Learning elastic constitutive material and damping models. In *Computer Graphics Forum*, volume 39, pages 81–91. Wiley Online Library, 2020.

[189] Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

[190] Kun Wang and WaiChing Sun. A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning. *Computer Methods in Applied Mechanics and Engineering*, 334:337–380, 2018.

[191] Tsun-Hsuan Wang, Pingchuan Ma, Andrew Everett Spielberg, Zhou Xian, Hao Zhang, Joshua B Tenenbaum, Daniela Rus, and Chuang Gan. Softzoo: A soft robot co-design benchmark for locomotion in diverse environments. In *The Eleventh International Conference on Learning Representations*, 2022.

[192] Tsun-Hsuan Johnson Wang, Juntian Zheng, Pingchuan Ma, Yilun Du, Byungchul Kim, Andrew Spielberg, Josh Tenenbaum, Chuang Gan, and Daniela Rus. DiffuseBot: Breeding soft robots with physics-augmented generative diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[193] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

[194] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[195] Mignon Wuestman, Jarno Hoekman, and Koen Frenken. A typology of scientific breakthroughs. *Quantitative Science Studies*, 1(3):1203–1222, 2020.

[196] 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. In *The Eleventh International Conference on Learning Representations*, 2022.

[197] Hongyi Xu and Jernej Barbič. Example-based damping design. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[198] Hongyi Xu, Funshing Sin, Yufeng Zhu, and Jernej Barbič. Nonlinear material design using principal stretches. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.

[199] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An end-to-end differentiable framework for contact-aware robot design. In *Robotics: Science and Systems*, 2021.

[200] Chao Xue, Xiaoxing Wang, Junchi Yan, Yonggang Hu, Xiaokang Yang, and Kewei Sun. Rethinking bi-level optimization in neural architecture search: A gibbs sampling perspective. In *AAAI Conference on Artificial Intelligence*, volume 35, pages 10551–10559, 2021.

[201] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

[202] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Conference on Neural Information Processing Systems*, 2023.

[203] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[204] 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, 2021.

[205] Naruki Yoshikawa, Kei Terayama, Masato Sumita, Teruki Homma, Kenta Oono, and Koji Tsuda. Population-based de novo molecule generation, using grammatical evolution. *Chemistry Letters*, 47(11):1431–1434, 2018.

[206] Yonghao Yue, Breannan Smith, Christopher Batty, Changxi Zheng, and Eitan Grinspun. Continuum foam: A material point method for shear-dependent flows. *ACM Transactions on Graphics (TOG)*, 34(5):1–20, 2015.

[207] Yonghao Yue, Breannan Smith, Peter Yichen Chen, Maytee Chantharayukhonthorn, Ken Kamrin, and Eitan Grinspun. Hybrid grains: Adaptive coupling of discrete and continuum simulations of granular media. *ACM Transactions on Graphics (TOG)*, 37(6):1–19, 2018.

[208] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.

[209] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International conference on machine learning*, pages 7444–7453. PMLR, 2019.

[210] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Efficiently programming large language models using sglang. *arXiv preprint arXiv:2312.07104*, 2023.

[211] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *international conference on machine learning*, pages 27042–27059. PMLR, 2022.

[212] Gengmo Zhou, Zhifeng Gao, Qiankun Ding, Hang Zheng, Hongteng Xu, Zhewei Wei, Linfeng Zhang, and Guolin Ke. Uni-mol: A universal 3d molecular representation learning framework. In *International Conference on Learning Representations*, 2023.

[213] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):10752, 2019.

[214] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, 2017. doi: 10.1109/ICCV.2017.244.